UNIVERSIDAD NACIONAL DE LA PLATA

FACULTAD DE INFORMÁTICA

TESINA DE LICENCIATURA

Título: Desarrollo modular en un simulador térmico satelital (STS)

Autor: Cacciagiú Georgina Gisele

Director: Ing. Luis Marrone

Asesor profesional: Ing. Mariana Solaro

Carrera: Licenciatura en Sistemas – Plan 2015

Resumen

El objetivo de la actual Tesina es desarrollar un simulador del modelo térmico satelital (STS) que utilice un núcleo de cálculo implementado como librería.

Realizar el estudio de los temas de investigación derivados en lo que refiere tanto a las técnicas de diseño y desarrollo de SW basado en librerías y la implementación del estándar SMP2 para codificación de simuladores.

Generar un núcleo de cálculo térmico con la posibilidad de reutilizar el código en futuras versiones de simuladores, apostando a la implementación modular de funcionalidades y con el objetivo de integrarlas a bajo costo.

Palabras Claves

Simulador Térmico Satelital.

Metodología de desarrollo COMET.

Estándar de Simulación SMP2.

Librerías Compartidas Linux.

Trabajos Realizados

Se estudiaron diferentes aspectos del estándar de simulación SMP2 y de interfaces entre distintos módulos existentes, para analizar, implementar y evaluar el desarrollo de un Simulador Térmico Satelital.

Conclusiones

Finalizado el desarrollo, se detectó que el uso de la herramienta como depurador del modelo térmico redujo las interacciones entre equipos y el re trabajo. Este proyecto afianzó mis conocimientos en sistemas de tiempo real, librerías compartidas Linux, estándares de simulación, etc. En cuanto a las estimaciones se concluyó que se sobredimensionarán las tareas menos precisas en futuros desarrollos. COMET como metodología dio resultados satisfactorios en el diseño de SW de tiempo real.

Trabajos Futuros

Representar el cálculo térmico de sistemas no satelitales.

Incluir SW de vuelo de control térmico y alarmas.

Incluir la librería térmica en simuladores de operación y de integración del satélite.

Representar el nivel de actividad de sensores y disipadores.

Guardar y recuperar un modelo térmico modificado.

Fecha de la presentación: Septiembre de 2017

Agradecimientos

Gracias a la Facultad de Informática de La Plata por esta oportunidad, por la formación que realizan a diario, orgullosa del proyecto que en mi generaron. A todos los profes y ayudantes que estuvieron en mi recorrido. A los amigos que encontré allí, no hubiera disfrutado tanto sin Uds. Gracias por su apoyo a diario y aun en la distancia, mi especial agradecimiento por dejarme hacer catarsis con la tesina. Los quiero.

A mi familia, porque además del apoyo económico, no dejaron que me rinda nunca, me acompañaron y me escucharon con mis problemas de correlatividades y promociones. Hoy hermanita soy feliz de escuchar tus dilemas y estoy muy orgullosa de vos. Inmenso gracias por su incondicionalidad.

Gracias Manu por hacerme notar que se podía lograr este objetivo, que no era difícil y que yo era capaz. Tu propósito es motivar a los demás, no me canso de decírtelo.

Gracias Gus y Marian por el apoyo para terminar mi carrera realizando un proyecto en el ámbito laboral, fue clave y oportuno. Infinitamente agradecida. También a mis compañeros de oficina, a Adrian y a Operaciones/Simulaciones por compartir sus conocimientos de simulación conmigo y por ayudarme en la concreción de este desarrollo. A Luis por guiarme en el progreso de la tesina. Todos son genios. Admiro el trabajo que hacen.

Gracias a mis amigos y familia barilochense que me ofrecieron su oreja para escuchar el log diario de la tesis.

Y finalmente quiero agradecer a mi abuela Dora por estar esos domingos preguntándome por la Facu mientras cocinaba, es el regalo más bonito de estudiar y de tenerla en mi vida.

Me siento afortunada por haber trabajado con mi viejo en el rubro (que a veces es difícil de describir a los de afuera) y por tener a mis abuelos para compartir este logro que llevó tantos años, los quiero Dora, Tata y Nene.

Contenidos

Índice

Ilustraciones

Introducción

Los satélites deben funcionar en un entorno físico muy diferente al terrestre por lo que es necesario que los elementos con los que están compuestas sus superficies, cableado, electrónica, etc. tengan un tratamiento especial para poder sobrellevar los fenómenos espaciales. Son ejemplos de estos fenómenos: el gran umbral de temperaturas al que se expone, las vibraciones excesivas que sufre el satélite en el lanzamiento, la falta de gravedad, la imposibilidad de cambiar piezas falladas una vez que está en vuelo, etc.

La forma más efectiva de desarrollar un satélite es realizando tantos modelos de Ingeniería como sean necesarios, donde se pueda realizar la integración de subsistemas y reemplazar las piezas que no funcionen como se espera. Para ello es necesario tener cierto stock de componentes de HW para no detener el avance de la Ingeniería.

En general, esto significa un gran costo para el proyecto. El tiempo de armado y prueba de los subsistemas implica un ciclo muy complejo, incluyendo además las demoras que pueden ocurrir en la recepción de los componentes, ya que muchos se compran en otros países.

La complejidad del problema aumenta en etapas más avanzadas del armado, donde existen puntos críticos en los que se verifica la consistencia del satélite. Estos puntos son las pruebas en el Shaker, las de sonido (que imitan los efectos del lanzamiento), y las de temperatura en las que los componentes pueden enviar de forma incorrecta sus datos a causa del ruido térmico.

En este tipo de procesos de Ingeniería es útil tener aproximaciones del comportamiento del sistema antes de que se encuentre armado completamente, y de esta manera poder corregir errores y prever problemas de integración.

Como solución a la falta de disponibilidad del Hardware (HW) y para reducir los daños que pueden ocurrir durante las pruebas funcionales y de integración, se comenzaron a utilizar los simuladores como herramientas de apoyo. Un simulador permite ensayar el comportamiento del sistema completo en determinado escenario y bajo ciertos estímulos operativos y del entorno, que van cambiando según avanza el tiempo en la simulación. Un escenario en particular puede ser el que representa al satélite en la zona de la órbita en la que se encuentra iluminado por el Sol, en ese momento los paneles solares producen potencia que se almacena en la batería. En este caso algunos heaters que son los encargados de calentar el satélite, no consumirán ni disiparán potencia ya que no es necesario mantenerlos encendidos porque el satélite se encuentra bajo incidencia del Sol. Otro escenario de interés específico de un satélite que toma imágenes radar es la adquisición de la imagen, la cual implica que la antena de radiofrecuencia este encendida consumiendo y disipando valores nominales, lo mismo para los otros subsistemas involucrados y en los casos de los subsistemas no involucrados (por ejemplo el canal de bajada de datos) debe considerarse su consumo y disipación en estado apagado, ya que el control de térmico y el perfil de consumo debe hacerse teniendo en cuenta todas estas relaciones entre las partes.

La ventaja que supone simular escenarios de interés es evaluar las particularidades positivas y negativas del uso de ciertos componentes en el armado de los subsistemas.

Este tipo de herramientas también son usadas para validar los procedimientos de operación, entrenar a los operadores y probar maniobras de contingencia ante fallas.

A estas principales ventajas se agrega el bajo costo de éstas herramientas, las mejoras que aportan para el testeo del sistema y la posibilidad de independizar el desarrollo de Software (SW) del de HW. Los simuladores son, por estos motivos, una ventaja estratégica para el área satelital.

Entre las funciones de un simulador de satélite se encuentran el envío de comandos, la inspección de telemetría, la inyección de fallas, la representación de la temperatura, entre otros. La fidelidad

del cálculo térmico es de suma importancia en este tipo de sistemas porque en el lanzamiento o en el espacio, el satélite estará en un ambiente extremo. Con rangos de temperatura muy amplios a causa de la radiación directa del sol: la cara que da al sol tiene temperaturas muy altas, mientras que la cara en sombra tiene temperaturas muy bajas. Esto no es admisible en el interior del satélite, donde se espera que la temperatura sea menos “extrema” y más estable, para no afectar el desempeño de la electrónica ya que estas variaciones están delimitadas por la tolerancia de la misma y de materiales de vuelo a la temperatura. Por lo tanto los errores en el cálculo de la temperatura pueden traducirse en una anomalía técnica, o la reducción drástica del tiempo de vida de la misión.

De esta forma surge la necesidad de una herramienta que facilite la validación de los cálculos del comportamiento térmico, dando lugar a la implementación del Simulador Térmico Satelital (STS) que muestre la variación de temperaturas en un ambiente equivalente al real.

La problemática térmica es recurrente en los simuladores y el modo en que se calcula la temperatura de los componentes aun tratándose de distintos modelos de satélites compuestos por distintos materiales. Bajo esta condición, se considera que la implementación del STS con la premisa de que use un módulo de cálculo térmico es un paso en la reutilización de código en el desarrollo de simuladores futuros.

Objetivo de la Tesina

i. Análisis de la situación actual

La solución que se elige para llevar adelante una misión en todas sus etapas de desarrollo es simular el comportamiento de un satélite, superando los inconvenientes de la obtención de los componentes.

Por ejemplo, los simuladores se emplean en la etapa de Ingeniería emulando el Hardware, para instalar el controlador de cada subsistema, y lograr ejecutar pruebas funcionales. El SW de vuelo de la plataforma del satélite también puede verificarse sin la necesidad de disponer de los componentes físicos. Desde el enfoque de la operación, facilita la verificación de procedimientos, el ensayo de maniobras especiales, el entrenamiento de los operadores usando aplicaciones de diagnóstico (visores de telemetría) sobre los datos simulados, etc.

Haciendo referencia al cálculo térmico la gran ventaja que propone, es que permitirá a los ingenieros térmicos probar sus algoritmos de propagación en un ambiente equivalente al real; a los ingenieros electrónicos detectar fallas en el diseño estructural o el de harness (cableado) de manera anticipada, a los desarrolladores de software de vuelo les permitirá chequear el correcto funcionamiento de los algoritmos de control térmico y alarmas que implementan para detección de errores.

El STS se presenta como una herramienta que aportará mejoras en la coordinación del avance del trabajo de los diferentes equipos involucrados y en la reducción de errores en etapas tardías de la Ingeniería del satélite. Plantear su desarrollo de manera modular permitirá una integración más simple del código relacionado al cálculo térmico a nuevos simuladores.

ii. Antecedentes en simulación

Un simulador del satélite facilita las pruebas con escenarios operacionales y la comprobación de los diferentes modos de funcionamiento. De esta forma el riesgo de realizar maniobras indebidas se reduce significativamente.

Antecedentes de simuladores utilizados en satélites argentinos en orden de creación son:

- Simuladores de satélite en la versión para ARSAT1 y ARSAT2.
- SSS (SAOCOM Satellite Simulator) para cada SAOCOM, que se está usando actualmente durante su Ingeniería para simular cada satélite, para realizar pruebas funcionales.
- el SCS (SAOCOM Constellation Simulator) que simula la constelación (grupo de satélites) de SAOCOM’s en su conjunto.

Estos simuladores tienen en común que implementan un estándar para representar modelos de simulación creado por la Agencia Espacial Europea, el cual se detalla más adelante en la sección Implementación de estándar: SMP2.

Los primeros simuladores de satélite, corresponden al ARSAT1 y ARSAT2 y se diferencian principalmente en los datos de configuración ya que debían reflejarse las modificaciones realizadas de un satélite a otro. Por ejemplo, el ARSAT2 posee 3 antenas para transportar señales de radiofrecuencia en banda Ku y banda C mientras que el primero sólo posee una de banda Ku, las posiciones geoestacionarias también son diferentes para ambos casos. Sin embargo los materiales que componen ambos satélites de telecomunicaciones, son los mismos. Mayormente están realizados en honeycomb por lo que el desempeño estructural frente a vibraciones, tracciones,

torsiones y tensiones durante la puesta en órbita es igual en ambos y la protección y el blindaje necesario a los equipos que los integran también, es decir que el comportamiento térmico de los materiales en consecuencia es similar. Cada satélite tiene 17 motores en total por lo que comparten este aspecto en cuanto a modelado de estructura y comportamiento [RD40].

En el simulador de SAOCOM a diferencia de los anteriores se debió tener en cuenta otros aspectos funcionales. El SAOCOM es un satélite Observación con Microondas es decir que su carga útil consta de un radar banda L en lugar de transponders de telecomunicaciones que presentaban los ARSAT. Tiene una órbita diferente, que es sol-sincrónica congelada con altura orbital de 620 km [RD41], la cual debió ser recalculada para poder simularla.

Las posiciones de despliegue si bien son las mismas (GEO, STOWED y GTO) en la estructura que posee el SAOCOM implica el movimiento de más piezas y mayores sectores de sombras entre partes de satélite ya que cuenta con una antena radar de 10mtrs. de largo; esto afecta tanto el comportamiento de la simulación de las áreas de sombra cuando el sol alumbra el satélite como las representaciones 3D que los operadores visualizan del satélite para identificar las distintas secciones del mismo.

Cabe mencionar que los avances arquitectónicos del SSS de SAOCOM respecto de los de ARSAT1 y 2 fueron relacionados a la modularidad, tal es así que se creó la librería de manejo de variables, de telemetría y comandos y de propagación de orbita: libcma.

Otro de los casos de simuladores desarrollados en la empresa es el simulador SCS que es monitor de un conjunto de simuladores de la constelación que se espera desarrollar para la misión SIASGE. Consta de dos grupos de satélites, uno conformado por satélites SAOCOM argentinos y otro por satélites COSMO SkyMed italianos. La disposición de los satélites en la órbita permite cartografiar globalmente la Tierra y que los paneles solares que posee, estén prácticamente siempre apuntando al Sol. De esta forma, integrando las capacidades de ambas constelaciones, se podrán obtener imágenes de cualquier catástrofe en cualquier punto del mundo, actualizadas cada 12 horas. INVAP desarrolla las constelaciones SAOCOM 1 y SAOCOM 2, cada una de las cuales consta a su vez de dos satélites A y B. Los satélites SAOCOM 1Ay 1B se encuentran en etapa de integración y ensayos en la Sede de INVAP en San Carlos de Bariloche, para ser lanzados a partir del año próximo [RD41]. En particular el simulador de esta misión debe tener control de las unidades SAOCOM, debe simular el ambiente y el segmento terreno que es el punto en la tierra desde donde se le envían los comandos cuando el centro de operaciones entra en la zona visible del mismo una vez al día.

Actualmente se encuentra en etapa de desarrollo la funcionalidad de integrar simuladores con HW real de manera que pueda reemplazarse el modelo simulado por el de vuelo o el de ingeniería.

Para el control y monitoreo de todos los simuladores, el SCS cuenta con una Base de Datos que contiene la lista de simuladores, su IP, Puerto y tipo del servidor en el que corren (Satélite, Estación Terrena, Ambiente). De esta manera el SCS puede obtener los datos de tiempo de cada simulador y controlarlos [RD3].

Los simuladores señalados son utilizados durante toda la vida útil del satélite:

- Durante su desarrollo: Los simuladores resultaron útiles para el desarrollo de SW de subsistemas que los componen.
- Durante la etapa de integración: Permitieron detectar problemas en la comunicación entre los componentes, problemas de endianness, de versiones de SW, entre otros.
- Durante el desarrollo de herramientas de soporte: Los visualizadores de telemetría de los diferentes subsistemas pudieron chequearse contra la funcionalidad provista por los SW de simulación.

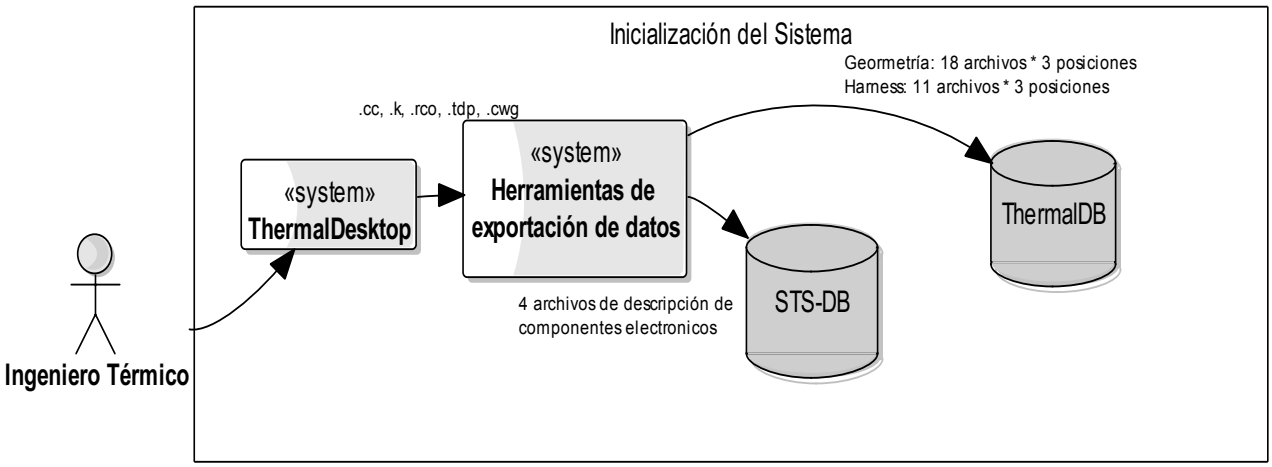
- Durante el desarrollo de los procedimientos de operación nominales: Se usaron los simuladores para verificar y depurar los procedimientos que permiten realizar las maniobras de despliegue, encendido, orientación, posicionamiento en órbita, siendo de mayor utilidad en los casos en que el HW no se encontraba disponible porque estaba siendo utilizado por otros equipos de trabajo.

- Durante el desarrollo de los procedimientos de operación no nominales: Luego del lanzamiento de los ARSAT se utilizaron los simuladores correspondientes para verificar el correcto funcionamiento de las operaciones extraordinarias a realizar ya sea para solucionar o dar contingencia a fallas de algún subsistema, sobrecalentamiento de componentes, etc. De esta manera se ensayan las maniobras antes de realizarlas en vuelo.

La necesidad de crear un simulador específicamente térmico (STS) surgió en base a los problemas de configuración del SSS.

Para simular el modelo térmico en el simulador de SAOCOM se requiere de una serie de archivos de configuración generados por el grupo de control térmico, que se los envía al grupo de simulaciones. En general, la integración del modelo térmico con el simulador de satélite requiere de varias iteraciones para la generación de estos archivos de entrada. Actualmente se definieron “buenas prácticas” para poder coordinar la exportación de datos de entrada desde un sistema a otro.

El siguiente diagrama muestra la cantidad de archivos .csv, que se utilizan para inicializar el sistema. Son 18 que describen la geometría del satélite y 11 que describen las relaciones entre nodos y las relaciones entre nodos y el espacio (29 archivos que se obtienen de la Herramienta de exportación de datos provenientes de ThermalDesktop, ambas herramientas quedan fuera del alcance de la tesina). Además es necesario realizar varios modelos térmicos según las configuraciones del satélite: imitando la disposición física durante el lanzamiento, cuando se coloca en órbita, cuando se despliega la antena, etc.



1 Arquitectura de Datos del STS

El hecho de que este proceso se realice manualmente, aumenta la posibilidad de fallas.

Por este motivo se reconoció la necesidad del STS como una herramienta de validación, que facilite la comunicación entre los diferentes grupos de trabajo, incluyendo a la empresa encargada del lanzamiento, quien también necesita conocer el modelo térmico del satélite.

Asimismo la implementación de un simulador que reutilice el modelo térmico como librería, es el siguiente paso en la reutilización de código para diferentes proyectos. Esta característica permitirá que la integración con simuladores futuros sea de manera modular y con un bajo costo.

iii. Alcance

Como resultado de esta tesina se implementó un simulador del modelo térmico satelital (STS) que utiliza un núcleo de cálculo implementado como librería.

Se estudiaron técnicas de diseño y desarrollo de aplicaciones basadas en librerías.

Se investigó también el estándar SMP2 para codificación de simuladores y su aplicación en forma de librería.

Se implementó el cálculo térmico como dll, para alcanzar las ventajas propias de la reutilización de SW.

iv. Motivación y Análisis del problema

Simulador Térmico

La empresa Invap SE se ha ganado un lugar de privilegio en el escenario internacional de la tecnología satelital.

Tiene como resultado de su vasta trayectoria como contratista principal la conclusión de varios sistemas satelitales: SAC-B, SAC-A, SAC-C SAC-D/Aquarius, ARSAT-1, ARSAT-2, además de los proyectos en curso como el Satélite Argentino de Observación Con Microondas (SAOCOM).

Los satélites son sistemas embebidos en los que la forma más efectiva de desarrollarlos es utilizando HW real. Sin embargo como se desarrolló anteriormente, existen muchos factores que hacen que esto no sea posible.

Se espera que la herramienta STS provea simplicidad al proceso de evolución del modelo térmico de satélites durante su diseño, permita validar los algoritmos de propagación de temperatura, validar el cableado (harness), la representación estructural, etc.

El STS se pondrá a prueba con datos del satélite SAOCOM, sin embargo a partir de la representación del modelo térmico de diferentes satélites podrán evaluarse las particularidades positivas y negativas del uso de ciertos componentes en el armado de los subsistemas.

Desarrollo Modular

El desarrollo modular permite, según el grado de avance del mismo, la evaluación del código desde diferentes niveles de abstracción. Así, puede hacerse testing unitario de las librerías que poseen la lógica del negocio. O bien, testing funcional de las librerías a través de su interfaz, verificando la funcionalidad de punta a punta.

En el caso particular de este desarrollo, las modificaciones en el cálculo de temperatura satelital, podrán realizarse sobre un único módulo, sin afectar el funcionamiento de aquellos programas que lo emplean.

Las librerías que conforman este proyecto, permiten la simulación de la temperatura de cualquier modelo satelital que se cargue en la base de datos. Podría modelarse la malla de nodos de un nuevo satélite que inicie la empresa, cargarse en la base de datos, e iniciar el simulador para validar sus relaciones sin aun disponer de telecomandos que le den instrucciones al satélite para llevarlo a un estado en particular. En el STS simplemente se configura el estado del satélite y el de su entorno; y puede observarse cómo se propaga la temperatura al iniciar la simulación.

El STS no está preparado para simular otro tipo sistemas embebidos, el motivo de ello es que se dispone de una serie de configuraciones dependientes del estado de despliegue que deben ser cargadas. Otro tipo de sistema puede carecer de piezas móviles y por lo tanto no presentar estados

de despliegue. Aunque esta situación puede representarse como un solo estado, este tipo de modelos quedan fuera del alcance de la tesina, pudiéndose plantear la necesidad como un trabajo futuro.

Organización del documento

En el Capítulo 1 se desarrollan conceptos técnicos específicos sobre desarrollo modular y una introducción al simulador térmico.

En el Capítulo 2 se estudian y proponen metodologías para arribar a la solución.

En el Capítulo 3 se realiza un análisis de herramientas técnicas empleadas en la implementación.

En el Capítulo 4 se presenta la estimación asignada al proceso de desarrollo y las memorias técnicas.

En el Capítulo 5 se repasan los pasos de instalación del software.

En el Capítulo 6 se analizan los resultados obtenidos, se plantean las conclusiones y posibles trabajos futuros.

Capítulo 1: Implementación modular y el STS

i. General

El simulador térmico se encarga de representar la evolución de la temperatura de un satélite y sus diferentes partes.

Este desarrollo se diseñó usando un estándar de simulación de la ESA (European Space Agency) como base para implementar modelos de simulación.

El STS es una entidad capaz de recibir llamadas a procedimientos y de producir información sobre la simulación como resultado.

Este SW puede operarse por medio de scripts, o mediante su interfaz gráfica. Ambas opciones de clientes, se comunican con la aplicación para el control y configuración de la simulación (iniciar, pausar y acelerar el tiempo, entre otros).

Gran parte de la funcionalidad del simulador es provista por el módulo térmico, que es el que se encarga de realizar los cálculos relacionados a las temperaturas en un período dado. Cuando la simulación inicia, comienza a correr el tiempo y el STS solicita al módulo que haga los cálculos necesarios para representar la variación térmica a la velocidad del tiempo configurada por simulador.

Este módulo tiene la capacidad de recrear el modelo térmico reducido del satélite a partir de datos cargados en una base de datos dedicada. Y tiene también la capacidad de parametrizar el entorno para poder observar los cambios de temperatura, dependiendo de la incidencia del Sol, la orientación de las piezas móviles del satélite, etc.

Un modelo térmico reducido se representa como un grafo, cuyos nodos se usan para identificar diferentes puntos significativos, que se relacionan entre sí y con el espacio disipando su potencia. Estos se encuentran conectados también a componentes eléctricos sensores, y disipadores de potencia, y a superficies que modifican la temperatura de su medio según sus propiedades o estado.

El módulo térmico fue pensado de manera que sea independiente al simulador, con el objetivo de reutilizar su código ya testeado, optimizar el uso de recursos y reducir el impacto de posibles cambios de diseño (de la BD y clases térmicas) o de la corrección de bugs en el cálculo térmico. Se espera que este módulo sea utilizado tanto en el STS como en futuros simuladores.

Este proyecto posee una arquitectura cliente-servidor que se comunican vía XML-RPC, donde el cliente puede ser una interfaz gráfica (implementada como librería con el nombre STSGUI) o un script python. El servidor consta de una librería que implementa el módulo térmico (libthermalcore), su interfaz (libthermalserverapi) y otra que implementa el simulador propiamente dicho (libstsmodels y libstsapi). Al tratar las partes de manera modular a través de librerías, fue necesario que cada una posea una interfaz bien definida para interactuar con el entorno en el que se encuentra instalada.

Bajo la arquitectura seleccionada para la implementación, el cliente es el que inicia la comunicación solicitando al servidor que ejecute cierto procedimiento. Luego el servidor consulta el modelo térmico por medio de las funciones que provee en su interfaz y devuelve el resultado de dicha operación al cliente.

Tanto el STS como la librería térmica tienen como requerimiento su implementación en Ubuntu 12.04 64 bits, y serán ejecutados en igual sistema operativo.

La arquitectura planteada se adapta a los recursos que este sistema operativo provee.

ii. Implementación Modular

Se espera que el STS haga los cálculos de la temperatura de un satélite teniendo en cuenta el paso del tiempo y la incidencia de distintos factores físicos.

Para resolver esta problemática es recomendable, estudiar la posibilidad de dividirla en otras más pequeñas, que puedan tratarse de manera aislada; para que, la complejidad global del problema disminuya considerablemente.

A cada subproblema se lo considera parte o módulo de la problemática global.

Dependiendo de las características de cada uno, se alcanzará un nivel de descomposición diferente, permitiendo abstraerse de la dificultad total para poder considerar por separado cada parte.

Los módulos que se desprenden de la problemática inicial (cálculo térmico y simulación de variables de entorno y tiempo) se interrelacionan entre sí, sin embargo, la manera en que cada uno de los módulos realice sus tareas no será visible para el resto, esto demuestra una de las principales características del desarrollo modular: la encapsulación. En el proyecto en cuestión se definieron interfaces de comunicación en las librerías libthermalserverapi y libstsapi a partir de las cuales se puede acceder a los servicios que ofrece la librería.

En resumen, la solución a un problema suele darse por un programa representado por un módulo principal (STS), el cual se descompone en otros módulos (calculo térmico, lectura de la base de datos, control de la simulación), los cuales, a su vez, también se pueden fraccionar, y así sucesivamente, es decir, el problema se resuelve por el método top-down [RD16].

Otra característica que se desprende de la definición de módulo, es la cohesión. Un diseño con una baja cohesión debería ser reformulado para que los componentes sean reestructurados en dos o más componentes pequeños [RD15]. Es decir que en una implementación modular cada subproblema debe resolver funciones relacionadas entre sí, lo cual nos facilitará su diseño, codificación, testing y mantenimiento. La forma de garantizar una fuerte cohesión es disminuyendo al mínimo las responsabilidades de cada módulo. En una implementación orientada a objetos este aspecto puede cuantificarse en una métrica que es usualmente calculada en clases bajo el nombre de Falta de Cohesión de Métodos (LCOM, Lack of COhesion of Methods). Existen diferentes variantes para medir la falta de cohesión LCOM1, LCOM2, LCOM3 y LCOM4 [RD15].

Uno de los objetivos de un desarrollo basado en la separación en subproblemas, es disminuir aquella relación entre módulos en donde los cambios sobre uno impactan sobre el otro, esto se conoce como acoplamiento. Un fuerte acople perjudica la reutilización de los módulos de forma aislada ya que uno depende del otro. Es decir que, cuando el acople es bajo, se incrementa la posibilidad de poder reusarlos por separado.

En referencia a conceptos de reutilización se puede mencionar que una arquitectura modular resulta en los siguientes beneficios:

- Escalabilidad: la estructura resultante es fácilmente adaptable a necesidades futuras. Permite la incorporación de nuevos elementos a cada módulo o nuevos módulos.
- Bajo acoplamiento: Los componentes de SW son independientes entre sí, lo que implica que las modificaciones en alguno de ellos no generan impacto en los demás.
- Uso del concepto de capas: las estructuras tanto del STS como del módulo encargado del cálculo térmico se desarrollaron para manejarse en dos capas de abstracción: una que entiende cómo son las relaciones en el modelo y otra que conoce los componentes eléctricos del satélite, las reglas de control y la máquina de estados de la simulación.

En los años ’60 se buscó dar solución a sistemas modulares a través de subrutinas. Estas se crearon para permitir que un bloque de código sea compartido llamándolo desde diferentes partes de un programa [RD43].
En la actualidad, es posible realizar sistemas de SW con arquitectura basada en componentes, que va más allá de la modularización del problema, buscando abstraer el módulo de la tecnología en el que se implementará. Existen también metodologías de desarrollo que detallan cómo esto puede ser llevado a cabo. Una de estas metodologías se detalla en la sección 3.i Proceso de Desarrollo COMET.

Cada componente es una unidad lógica de distribución, reutilizable que se comunica con el entorno por medio de mensajes definiendo una interface para ello. Para implementar estos componentes es posible utilizar librerías compartidas de Linux, con las que se implementaron el STS y la LibThermalCore. Las librerías cumplen con el principio de modularidad, de forma que si se desea reutilizar código o si se necesita modificar alguna funcionalidad (que no cambie la interface), bastará con editar dicha la librería, compilarla e instalarla, sin necesidad de modificar el programa que las utiliza. Se detallan estos temas en la sección “Desarrollo Modular a través de Librerías”.

iii. Simulador térmico

Los simuladores presentan ventajas estratégicas para los desarrollos realizados en el área satelital, ya sea en la integración de los subsistemas que componen la plataforma o en la carga útil (radares, cámaras, transponders).

Entre las funcionalidades que incluyen estas herramientas cada vez más utilizadas existe: la emulación de piezas de HW, SW, aviónica, física del satélite, recepción de comandos, inserción de errores, manejo de la fluctuación de la temperatura, etc.

El origen de esta tesina basada en la realización de un SW que simule exclusivamente cálculos de temperatura a través del paso del tiempo, de manera independiente a otros simuladores que se encargan de la funcionalidad total, surgió a partir de una necesidad de mejora del procedimiento de trabajo como se explica a continuación.

La mayor dificultad respecto a la simulación de esta funcionalidad es que necesita un modelo térmico reducido como parámetro de entrada en las ecuaciones de cálculo.

El modelo térmico es específico de cada satélite. Si bien se reutiliza gran parte del modelo de un satélite a otro, se debe tener en cuenta si se modificaron componentes eléctricos, mecánicos o materiales, es decir, debe representar de manera abstracta las características de trasmisión de temperatura entre las distintas piezas.

Para lograr esto se define el modelo como un multigrafo dirigido [RD19], donde:

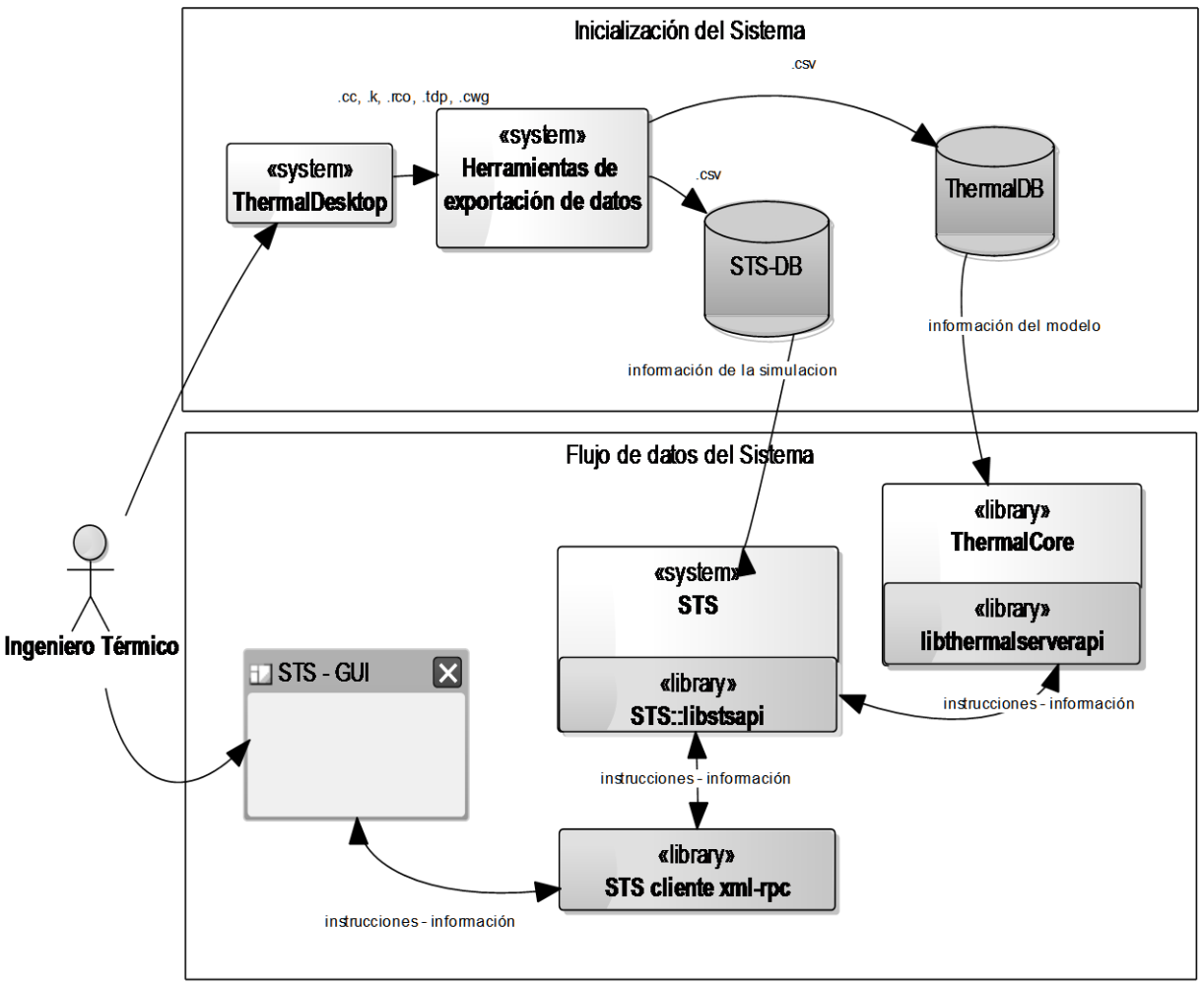
- los nodos son elementos que pueden calentarse, los cuales se definen por una masa y capacidad calorífica
- las aristas representan la trasmisión de temperatura entre los nodos por conductividad (temperatura trasmitida por contacto), por radiatividad en las caras internas y externas del satélite (el grado de visión entre un nodo y otro es la relación calorífica entre ellos en el vacío).
- se representa con un coeficiente la disipación de potencia del nodo al espacio profundo
- además se relacionan los nodos con componentes eléctricos que disipan potencia o censan temperatura y con superficies que pueden estar total o parcialmente expuestas al Sol y por lo tanto pueden calentarse o necesitar que sea calentada por un mecanismo de control térmico.

Así el cálculo térmico puede entenderse como una resolución de ecuaciones diferenciales [RD20]. El método de resolución seleccionado es confidencial.

Quienes se encargan de elaborar este modelo son los integrantes del equipo térmico. Estos ingenieros son responsables de la creación de un modelo térmico reducido y representativo del satélite. Para realizar tal trabajo operan en ThermalDesktop realizando los cálculos necesarios y exportan sus datos a archivos que servirán de información de entrada en la base de datos de cualquiera de los simuladores existentes que contemplan la temperatura de un satélite, como por ejemplo el simulador de satélite de SAOCOM 1A. En esta relación entre el equipo de trabajo que genera el modelo y el equipo encargado de desarrollar los simuladores, es necesario un paso intermedio de depuración del modelo térmico entregado. De manera que los archivos de entrada que se utilicen sean completos y correctos antes de incorporarlos en un entorno de alcance mayor. Generalmente un error en el diseño del modelo, produce valores inconsistentes durante la simulación; esto se puede notar al observar cómo la temperatura de todos los nodos tiende a infinito en cuestión de segundos.

Esta falencia en el procedimiento en la que se distingue la falta de un paso de depuración del modelo, generó la búsqueda de un simulador dedicado, que permita al equipo térmico, diseñar el modelo y poder verificar que con el correr del tiempo en la simulación, los resultados son consistentes.

Este es el motivo por el que se implementó el STS, que permite pasar a la etapa de integración (modelo térmico-simulador de satélite completo), con archivos de configuración válidos, reduciendo el tiempo de interacción y coordinación entre distintas áreas.



2 Módulos STS

En el diagrama de arquitectura de datos del STS se muestra el flujo de los mismos desde el momento en que se generan los archivos de configuración en el ThermalDesktop. Luego los transforman con la herramienta TD2DSS (ThermalDesktop to DSS) a un formato de entrada especifico (.csv) de manera que sirvan para la creación de la base de datos del simulador térmico (compatibles también con el SSS). Obtenidos estos archivos, se pueden ejecutar unos scripts que cargan la base de datos y a continuación la aplicación STS (que incluye el servidor xml-rpc receptor de pedidos de algún cliente). Luego puede ejecutarse la interfaz STSGUI que permite la interacción del usuario con el servidor para simular la evolución de los nodos térmicos.

Asimismo el STS permite probar cambios en el modelo térmico cargado en memoria y chequear su comportamiento en el tiempo, sin necesidad hacer la modificación de sus archivos de configuración en cada propuesta.

Este nuevo SW permite también reproducir casos de fallas en las que algún componente toma una temperatura constante, ya sea porque se sobrecalentó o dejó de funcionar, o focalizarse sólo en los valores que toma un subsistema filtrando por algún criterio. Además se puede configurar el ambiente en el que vive ese satélite, y la posición de sus superficies móviles según el estado de despliegue. Facilita igualmente, el análisis teniendo en cuenta el envejecimiento de los materiales.

iv. Reusabilidad del STS y la LibThermalCore

Para cada proyecto satelital que se diseña en la empresa se suelen implementar nuevas aplicaciones de simulación. Por este motivo es necesario contar con reutilización de código de manera que el desarrollo sea más eficiente en los casos en que el proyecto es similar al anterior, es el mismo con una configuración diferente, o posee una nueva arquitectura con elementos existentes. En particular en estos simuladores existen ciertas funcionalidades comunes entre cada nueva versión: los controles de la simulación, la configuración del entorno/espacio, la evolución de la temperatura, la definición de estados de despliegue, etc.

Es importante en este sentido comprender la arquitectura del código original para identificar los componentes, los límites y las interfaces, determinando qué es potencialmente reutilizable. El modelaje en UML es valioso en este proceso porque ayuda a analizar el código y a tomar decisiones acerca de la modularización.

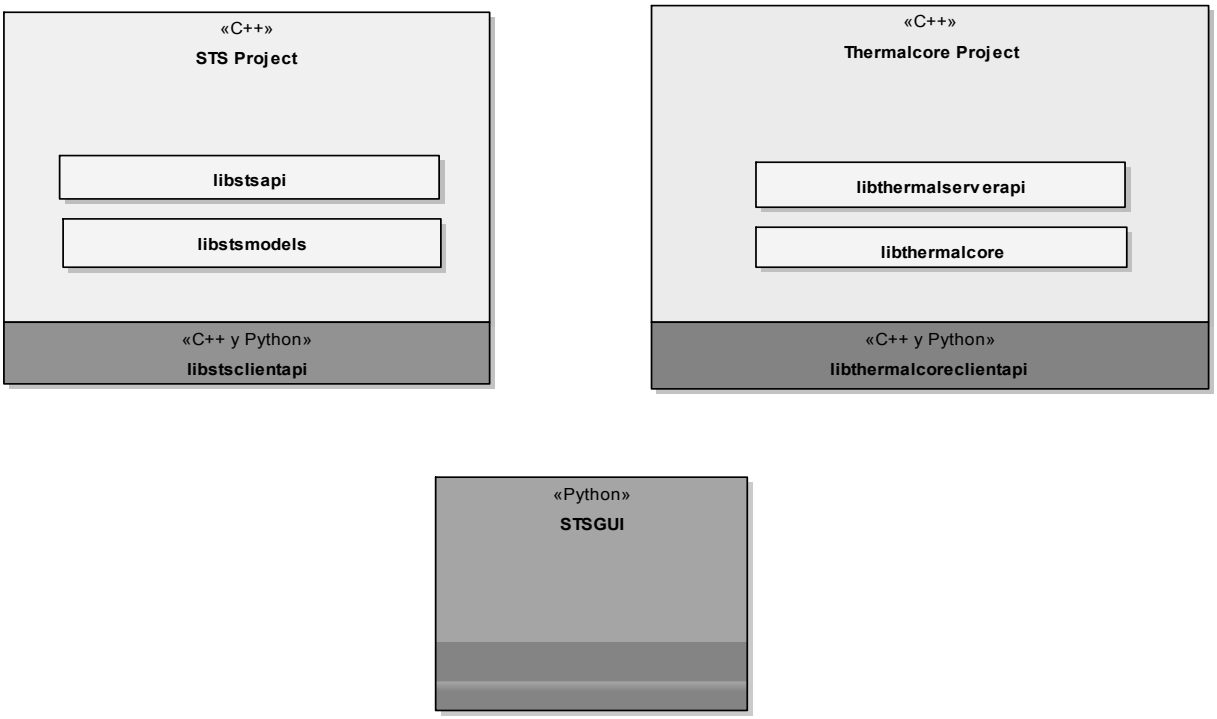
Meyer considera que la reutilización de software tiene costos asociados [RD22]:

- Costo de aprendizaje (al menos la primera vez).
- Costo de integración en el sistema. En particular si es preciso escribir adaptadores para conectar adecuadamente el componente en el sistema que se está desarrollando.

Griss (1993) aconseja el desarrollo de modelos de cálculo del ROI (Return Of Investment, retorno de la inversión) para convencer a la dirección de la conveniencia de tratar la reutilización como un activo importante [RD22].

El concepto de librería se ajusta bastante bien a la idea de reutilización ya que agrupa un conjunto cohesionado de funciones, en particular las librerías de enlace dinámico (DLL) que permiten que dos programas que requieran usar la misma función no tengan el código duplicado, sino que compartan un único ejemplar de la librería.

Un primer paso que se dio fue la implementación de una librería que permita controlar la simulación implementando el estándar SMP2, utilizada hoy en muchos de los desarrollos que acompañaron las campañas de satélites anteriores. En base a esta experiencia positiva se decidió realizar la implementación de otras funcionalidades en librerías, entre ellas una que reproduzca la evolución de la temperatura en el tiempo y un simulador stand alone que posibilite depurar el modelo térmico antes de incluirlo en un simulador de funcionalidad global, dando lugar a la LibThermalCore y al STS.



3 Estructura

En el gráfico se muestra la estructura de la librería térmica (libthermalcore), el STS (libstsmodels), su interfaz gráfica (STSGUI) y las librerías de comunicación xml-rpc cliente (libstsclientapi, libthermalcoreclienteapi) y servidor (libstsapi, libthermalserverapi).

Estos desarrollos se realizaron durante el periodo de Ingeniería del SAOCOM por lo que la definición de los datos surgió de su modelo térmico reducido. Sin embargo el objetivo final de la implementación fue realizar un simulador que pueda ser usado para cualquier modelo térmico cambiando los datos de configuración, esto significa que el STS podría mostrar la evolución de las temperaturas en el tiempo y en el entorno físico del espacio, para los sucesivos satélites que se desarrollen en la empresa.

La librería térmica como colección de componentes reutilizables relativos a un dominio, encapsula un conjunto de funciones que están semánticamente relacionadas, siendo de esta manera: modular y cohesiva.

El modo en que este componente se comunica con los simuladores es por medio de su interface (libthermalserverapi). Esta interfaz especifica los servicios que otros componentes pueden utilizar, y cómo pueden hacerlo sin necesidad de exponer el funcionamiento interno de la misma.

Es importante que librerías de clases como ésta posean ciertas características para su reutilización: completitud, adaptabilidad, eficiencia, seguridad respecto a los tipos, simplicidad y extensibilidad [RD21].

Apoyándose en los principios de abstracción, encapsulamiento, modularidad y jerarquía es posible lograr un desarrollo que pueda ser usado en varios componentes de SW.

Es necesario destacar que la principal ventaja que se presenta al generar una librería de código, es la posibilidad de testear dicha funcionalidad por única vez y, al incluirla en otro desarrollo, garantizar que las funciones térmicas están verificadas, es decir que se incrementa la fiabilidad, reduciendo el esfuerzo, el costo y los plazos de ejecución del nuevo desarrollo tanto en testing como en codificación, aumentando así, la productividad. Del mismo modo, de ser necesarios cambios en el modelo térmico o en el cálculo, y siempre que se mantenga la definición de la API original; solo será necesario el testing sobre ese componente de manera aislada, lo que conlleva una disminución de los costos de mantenimiento.

Capítulo 2: Metodología de desarrollo

i. Proceso de desarrollo COMET

Para este proyecto de simulador se escogió un método de diseño de software basado en UML (Unified Model Language). COMET (Concurrent Object Modeling and Architectural Design Method) es una metodología de desarrollo de software OO (Object Oriented) con un ciclo de vida altamente iterativo, que aborda las fases de modelado de requerimientos, análisis, y diseño.

Describe métodos y técnicas que hacen posible el uso de componentes y objetos de negocio como bloques constructores de sistemas de SW dinámicos y flexibles, fáciles de mantener y de evolucionar.

Esta metodología recomienda la combinación de desarrollo iterativo y de desarrollo incremental. De manera iterativa, se obtiene el estado inicial de los requerimientos con un modelo de Caso de Uso, luego se selecciona un pequeño grupo de Casos de Uso y el sistema se diseña e implementa mediante incrementos que cumplimenten esas funcionalidades seleccionadas. Cada incremento debe generar un sistema usable. Estos pasos: seleccionar, diseñar e implementar Casos de Uso, se realizarán hasta completar todos los escenarios que en ellos se proponen.

Con una amplia especificación de pasos según el tipo de sistema a desarrollar, COMET es una práctica guía que indica cómo analizar, diseñar e implementar sistemas, basándose en objetos de negocio.

Según el consorcio de estándares de tecnología Object Management Group (OMG) [RD7] “el modelado es el diseño de aplicaciones de SW antes de la codificación”. Utilizando un modelo, pueden asegurarse que la funcionalidad del negocio es completa y correcta, las necesidades del usuario final son satisfechas y el diseño del programa soporta los requerimientos de escalabilidad, robustez, seguridad, extensibilidad y otras características antes de su implementación.

UML ayuda a especificar, visualizar y documentar modelos. Es una notación gráfica para describir los resultados de un análisis OO y es independiente de la metodología. Los principales motivos por los que COMET utiliza UML como representación gráfica, son su aceptación internacional como un estándar para expresar análisis y diseño orientado a objetos, puede usarse en muchas herramientas abarcando una gran cantidad de usuarios.

Un modelo UML puede ser independiente de la plataforma (PIM Platform Independent Model) o específico de una plataforma en particular (PSM Platform Specific Model). La ventaja de manejar modelos PIM, es que permite reusabilidad de los mismos cualquiera sea la tecnología subyacente. El enfoque que plantea COMET se alinea con OMG que sugiere el uso de MDA (Model Driven Architecture) para desarrollar SW basado en componentes expresados en modelos UML PIM.

La arquitectura orientada a modelos descriptos en UML provee conceptos básicos para representar una arquitectura independiente de su plataforma y también para modelos específicos de la plataforma.

La selección de la metodología COMET surgió de la comparación con otras que consideraran la generación de pequeños entregables, diseño de componentes concurrentes y real time, y que empleen preferentemente notación gráfica UML. En particular COMET se destaca por considerar en la etapa de diseño, el mapeo del modelo de análisis a un modelo de diseño concurrente. También respecto de las actividades de las etapas de construcción incremental, considera el testing de integración como actividad del equipo de desarrollo dejando a cargo del equipo de testing las pruebas de sistema [RD8]. Incluye también entre sus actividades la validación temporal para los casos de sistemas de tiempo real a través del análisis de secuencias de eventos. Esta capacidad de COMET es destacada en el estudio comparativo de conceptos de tiempo real entre

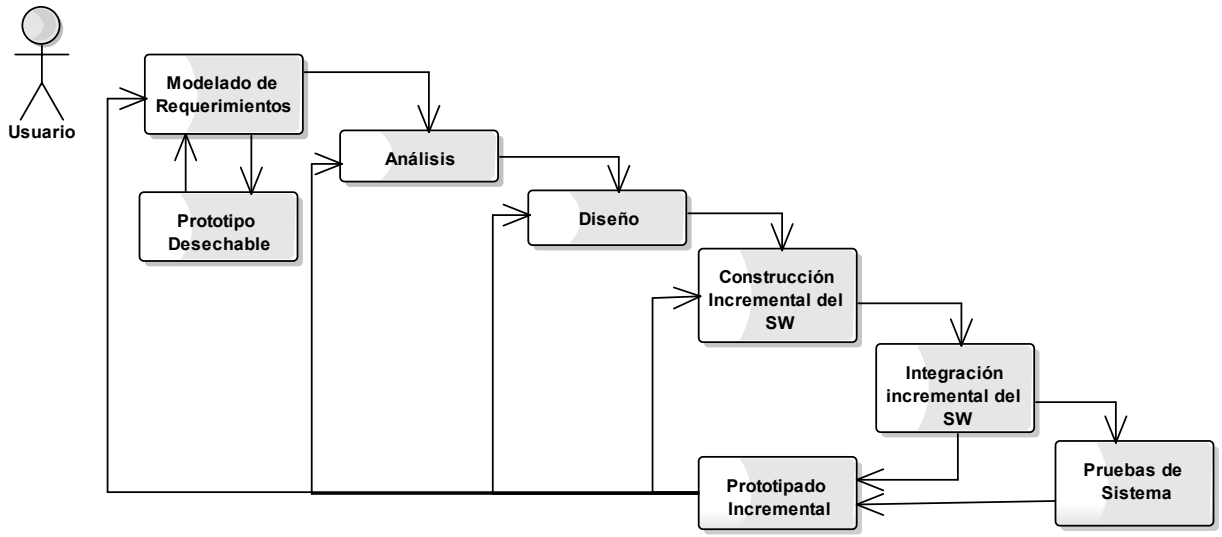
COMET, ROPES y OCTOPUS [RD6]. En esta publicación “Se resalta la importancia de aplicar una metodología en el desarrollo de Sistemas de Tiempo Real y los beneficios que ésta puede aportar al producto final si se combina con otros conceptos de la ingeniería del software.” En este estudio, para comparar, consideraron que las metodologías posean documentos de describan concurrencia, manejo de eventos, comunicación por mensajes y validación temporal. En este último punto indicaron que sobresale COMET al considerar los diagramas de secuencia en el tiempo, mientras que ROPES y Octopus/UML no ofrecen ningún modelo para la validación temporal.

Por otra parte, comparando COMET con Proceso de Desarrollo de SW Unificado (USDP) según [RD8], se pudo observar que comparten el lenguaje de modelado UML y las primeras 3 etapas de su ciclo de vida. Sin embargo COMET considera arquitecturas concurrentes en su etapa de diseño y agrega, tanto en el modelado de requerimientos como en la construcción incremental, la posibilidad de crear prototipos desechables que ayuden a validar los requerimientos y reducir riesgos. COMET desacopla la fase de testing del flujo de trabajo de USDP en dos etapas: la integración incremental del SW y los test del sistema. La primera es una actividad del equipo de desarrollo y la segunda del equipo de testing como se mencionó anteriormente.

Respecto de Metodologías Agiles, en particular Scrum, la forma de trabajo que propone, es altamente iterativa e incremental pero no provee actividades de diseño específicas para cada tipo de arquitectura, en el caso de COMET como se describirá más adelante en esta sección, tiene en cuenta y detalla arquitecturas cliente-servidor, envíos de mensaje en arquitecturas distribuidas de control centralizado o distribuido, diseño de tareas concurrentes, diseño de sistemas real time, entre otras.

Hassan Gomma en [RD8] propone el uso del Modelo de Espiral en conjunto con COMET. Durante la planificación del proyecto en un determinado ciclo del espiral se decide qué tarea técnica se realizará en el cuadrante de desarrollo de producto (III Cuadrante del Modelo de Espiral). La actividad seleccionada puede variar entre las mencionadas fases de COMET (modelado de requerimientos, análisis o diseño). El análisis de riesgos del Cuadrante II y la planificación del ciclo del Cuadrante IV indican cuántas iteraciones serán necesarias para esas actividades.

A continuación se detallarán las diferentes etapas según el ciclo de vida del SW de COMET que se describen en este gráfico:



4 Ciclo de vida del SW

Durante el modelado de requerimientos, los requerimientos funcionales del sistema se definen, en términos de actores y Casos de Uso. Cada Caso de Uso (CU) define una secuencia de interacciones entre uno o más actores y el sistema. Esto permite especificar la funcionalidad desde diferentes niveles de detalle. Un prototipo desechable en esta etapa ayuda a esclarecer las necesidades de los Stakeholders.

En la etapa de análisis, se hace énfasis en comprender el problema en términos de objetos y mensajes entre ellos, se realizan modelos estáticos a través de diagramas de clase que ayudan a visualizar las relaciones entre las clases involucradas en el sistema y, modelos dinámicos que muestran la participación de los objetos en los Casos de Uso, se usan para ello diagramas de comunicación, diagramas de estado que representan cambios en el sistema ante determinados sucesos, y diagramas de secuencia para modelar interacción entre objetos.

En la etapa de diseño, se desarrolla la arquitectura del SW y el modelo que se obtuvo como resultado del análisis se mapea al dominio de la solución, se identifican y estructuran los subsistemas, haciéndose énfasis en el diseño de subsistemas distribuidos como componentes configurables que se comunican a través de mensajes. Se busca una solución a los problemas de distribución, concurrencia y ocultamiento de información. Se emplean diagramas UML tales como:

- Diagramas de contexto para definir la arquitectura estática del SW mostrando su relación con otros sistemas o en un nivel más detallado, su dependencia con otras librerías
- Diagramas de interfaces para mostrar encapsulamiento de información
- Diagramas de componentes para visualizar la estructura de alto nivel del sistema y el comportamiento del servicio que estos componentes proporcionan y usan a través de interfaces

Luego de finalizar el diseño, tiene lugar la construcción incremental del sistema. Esta se basa en la selección de un subconjunto de funcionalidades para ser construidas en esa iteración. Implica un diseño más detallado, codificación y testing. Se utilizan diagramas de secuencia para modelar la interacción entre objetos del dominio y diagramas de tiempo que muestran el cambio de valor de elementos en un período, la interacción entre los eventos y sus restricciones. Además pueden explotarse otros diagramas ya mencionados para obtener mayor nivel de detalle.

Se complementa esta etapa mediante la integración del nuevo incremento al sistema. Se realiza testing de integración basado en las funcionalidades seleccionadas para esa iteración. Estas pruebas chequean la interfaz de los objetos que participan en los Casos de Usos seleccionados.

Cada incremento de SW forma el prototipo incremental. En este punto se itera las veces que sean necesarias para lograr el refinamiento de las especificaciones de SW, su análisis y diseño. Una vez alcanzada la versión final, se generan casos de prueba de tipo caja negra, a partir de los Casos de Uso.

A continuación se describe con más detalle cada una de las etapas realizadas en este proyecto.

Modelado de Requerimientos

Las actividades según COMET para esta fase son el modelado de Casos de Uso y la definición de requerimientos no funcionales que pueden no estar representados en UML.

Durante esta fase, se indagó en el dominio por medio de entrevistas a los distintos Stakeholders. En una primera iteración el análisis apuntó a conocer los requerimientos de alto nivel que permitieran identificar los límites del sistema, éstos pueden ser la interacción con la interfaz gráfica, los resultados que se esperan que el STS provea o la versatilidad que debería ofrecer la publicación de la funcionalidad térmica en una librería.

Según COMET, es necesario tener presente el cambio que se desea alcanzar con el nuevo desarrollo, en este caso como se mencionó al comienzo de la tesina, se desea mejorar el flujo de trabajo y la interacción entre el equipo de ingenieros térmicos y el grupo de desarrollo de SW de simulación, por lo que se cree que la existencia de un simulador que los térmicos puedan operar permitirá pulir su modelo antes de interactuar con el otro equipo.

Además debe hacerse una primera aproximación del Análisis de Riesgos que posibilitará entender la criticidad de algunas tareas y prever casos de contingencias.

Algunos riesgos identificados fueron:

RIESGOS	PROBABILIDAD[1]	IMPACTO[2]	ESTRATEGIA[3]
1. Subestimación del tamaño del sistema	Alto	Serio	Disminuir: *Tomar decisiones comparando el sistema a realizar con SW similares en producción*
2. Subestimación de la cantidad de componentes a implementar	Alto	Tolerable	Disminuir: *Realizar un análisis que tenga una visión general del dominio*
3. Falta de compatibilidad entre el producto a generar con las librerías que tiene como dependencias	Bajo	Serio	Anular: *Utilizar la arquitectura necesaria para poder ser compatible con las dependencias*
4. Falta de disponibilidad del entorno de desarrollo configurado o de herramientas de gestión del proyecto	Muy alto	Tolerable	Disminuir: *Solicitar herramientas, versiones compatibles de las mismas o manuales de configuración de ambiente a quienes hayan trabajado en desarrollos similares*
5. Finalización fuera de plazos originales	Moderado	Serio	Disminuir: *Realizar una planificación, asignar más recursos y eliminar los obstáculos que puedan surgir para facilitar el progreso del proyecto*
6. La detección de errores de arquitectura que impliquen volver a las fases de diseño	Moderado	Insignificante	Disminuir: *Utilizar un ciclo de vida iterativo e incremental que permita dar pasos pequeños en la implementación e ir testeando a medida que se integran las nuevas funcionalidades*
7. Subestimación del tiempo requerido para cada una de las fases del proceso	Moderado	Tolerable	Transferir: *Consultar estimaciones para las fases con alguien que tenga experiencia en desarrollos similares*

[1] Se consideran los siguientes parámetros para clasificar los riesgos según probabilidad: (muy bajo (<10%), bajo (10-25%), moderado (25-50%), alto (50-75%) o muy alto (>75%)).

[2] Se consideran las siguientes categorías para clasificar los riesgos según el impacto: catastrófico, serio, tolerable o insignificante

[3] Aceptar si el riesgo es pequeño, Transferir a otro responsable, Anular la probabilidad del que el riesgo ocurra o Disminuir el impacto.

8. Falta de disponibilidad del cliente para reuniones con el equipo de desarrollo	Muy bajo	Tolerable	Aceptar
9. Subestimación de la curva de aprendizaje de las nuevas tecnologías a utilizar	Alto	Serio	Transferir: *Consultar nivel de dificultad y material de capacitación con especialistas en las nuevas tecnologías*

5 Riesgos identificados para el proyecto STS

	Insignificante	**Tolerable**	**Serio**	**Catastrófico**
Muy alto		4		
Alto		2	1, 9	
Moderado	6	7	5	
Bajo			3	
Muy bajo		8		

6 Priorización de los riesgos del STS

Las bandas de color indican una forma de priorizar los riesgos: en rojo son los más críticos, en verde los menos.

Es necesario también identificar los principales recursos del negocio, por ejemplo en el STS son recursos importantes el desarrollador, programadores expertos en desarrollos similares, las librerías a reutilizar, herramientas de apoyo y de entrada de datos (como las herramientas de exportación de datos desde ThermalDesktop a csv), herramientas para testing unitario.

Para complementar el análisis de requerimientos es de importancia identificar qué actividades se realizan de manera completamente manual, cuáles con apoyo de una herramienta y cuáles son automatizadas por el SW a desarrollar, de esta manera se entiende la visión de cambio que ofrece el producto y se identifican los riesgos del actual escenario. Por ejemplo la creación de archivos de configuración del modelo y la carga de la base de datos se realiza mediante scripts de soporte. La verificación del modelo térmico antes del STS, debía hacerse integrándolo a un simulador de satélite. La visión de cambio que ofrece el STS es permitir que el debug del modelo térmico se haga de forma previa a la integración con un simulador de mayor alcance para reducir la sobrecarga de trabajo del equipo de desarrollo de simuladores.

Así, el conocimiento de la realidad operativa, ayuda a plasmar la información en una descripción narrativa de la interacción usuario-sistema dentro de los Casos de Uso. En efecto, se logró avanzar un poco más en la comprensión del problema de manera top down, por lo que, en las siguientes iteraciones, se investigó sobre cuestiones técnicas relacionadas a la composición del modelo térmico, el cálculo de las temperaturas, la identificación de los sistemas que a futuro utilizarán la librería, el diseño de simuladores existentes, los estándares y herramientas que utilizan.

La participación activa del Product Owner en la recopilación de información fue esencial en esta etapa. En particular con el STS no fue necesario realizar un prototipo desechable para comprender la totalidad de los requerimientos. Esto fue posible gracias a los conocimientos adquiridos sobre simuladores anteriores, las experiencias y know how alcanzados en desarrollos aún más complejos como es el caso de los simuladores de Arsat1, Arsat2, SAOCOM y el SCS [RD1][RD2][RD3] mencionados en la sección Antecedentes en Simulación.

La documentación generada con los Casos de Uso, cumplimenta el primer requerimiento de validación, que indica que la funcionalidad debía ser corroborada con los Stakeholders.

Se tuvieron en cuenta las necesidades funcionales y no funcionales de los usuarios, entre estas últimas se encuentran los requerimientos de rendimiento, los de validación y las restricciones de diseño e implementación.

Además se identificaron los entregables a realizar para dimensionar el alcance y el plan de desarrollo del SW. Este tema se detalla más adelante en el Capítulo 4.

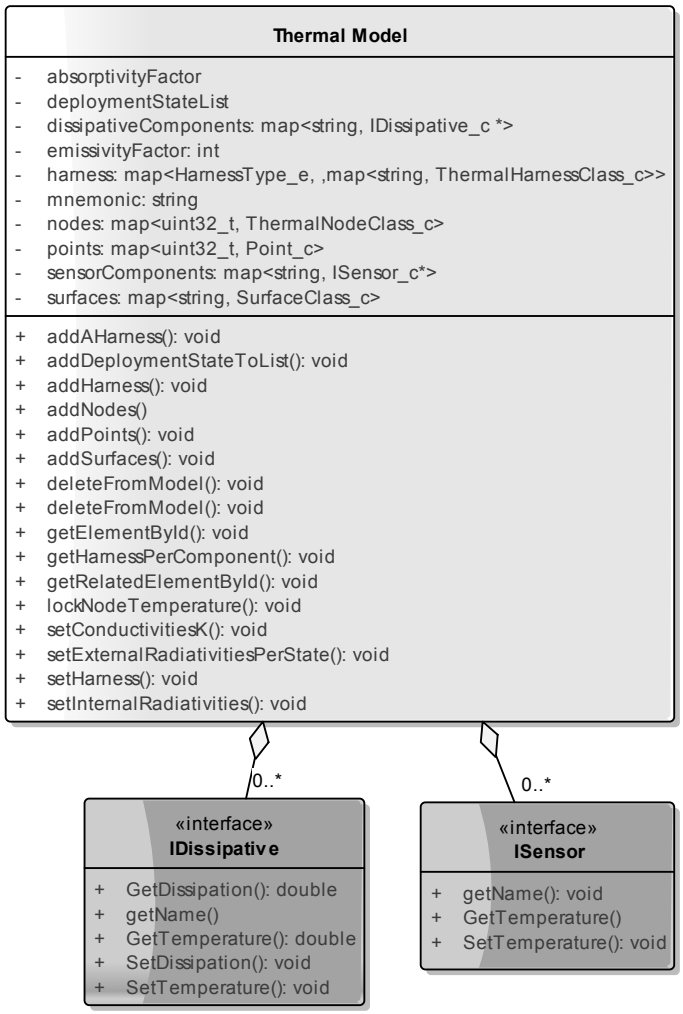
La documentación que refleja el trabajo de esta etapa del ciclo, es la Especificación de Requerimientos que se puede observar en detalle en el Apéndice A.

Análisis

En la fase de análisis se desarrollan los modelos estáticos y dinámicos del sistema.

El estático define las relaciones estructurales entre las clases del dominio del problema a través de diagramas de clases.

Se realizaron algunas iteraciones sobre los diagramas de clases a medida que se iba comprendiendo el dominio. Tal es el caso del refactoring que se realizó en el diseño de las clases térmicas donde se identificó que los termistores, termostatos y termocuplas tenían igual comportamiento y composición desde el punto de vista térmico (temperatura y nombre). Por este motivo se los agrupó bajo la interfaz ISensor. Por otro lado, las cargas y heaters con la capacidad de conocer la potencia que disipan, su temperatura y nombre, se agruparon bajo la interfaz IDissipatives. El conocimiento en detalle del problema dio la posibilidad de abstraerse de los pormenores del mismo.



7 Simplificación del modelo térmico

El modelo dinámico se encarga de describir qué objetos participan en los Casos de Uso y cómo interactúan entre sí, a través de los diagramas de comunicación o de secuencia, o bien si el objeto posee estados, estos pueden detallarse mediante diagramas de estados. En esta etapa se evaluaron los tipos de mensaje que las librerías se envían, los estados por los que pasa el simulador, etc. Estos datos se detallan en el Apéndice B – Descripción del Diseño del SW.

Entre las actividades de esta fase se deberían organizar los Casos de Uso en subsistemas, separando la funcionalidad del simulador y de la librería térmica. En este punto se identificó que tanto para la librería térmica como para el simulador debían existir otras dos librerías, una librería que registrara la API específica en el servidor y una que permita consumir estos métodos desde un cliente.

Se consideraron los aspectos dinámicos del sistema según explica Hassan Gomma. En el modelo de análisis, el sistema debe representarse como una colección de objetos colaboradores que se comunican por medio de mensajes. Se analizan los objetos para determinar cuáles se pueden ejecutar simultáneamente, cuáles necesitan ejecutarse secuencialmente y así se determina cuál debe ser activo o pasivo. De aquí se consideró la existencia de un servidor que controle los cambios de estados y una interfaz gráfica que le dé instrucciones del usuario.

Asimismo fue necesario empezar a derivar las interfaces entre los componentes y sus colaboraciones a través de la especificación de operaciones, su protocolo y semántica. Iniciando por el análisis de los escenarios de los CU, se pensaron los métodos con los que las librerías debían comunicarse con su contexto, dando como resultado los que se detallan en el Apéndice D – Documento de Referencia de la API del STS.

Fue necesario también iterar en estos pasos para refinar la arquitectura y reducir los riesgos. Con tal fin se realizaron varias reuniones con los Stakeholders y con desarrolladores con la experiencia de otros simuladores.

Diseño

En esta fase el modelo de análisis se mapea a un modelo de diseño.

En el diseño de arquitectura:

- Se desarrollan diagramas de comunicación de objetos.
- Se toman decisiones sobre las estructuras e interfaces de los subsistemas
- Se desarrolla la arquitectura general del SW
- Se decide qué patrones arquitectónicos y de diseño se usarán, cómo serán las interfaces de las clases y cómo ocultarán su información interna.
- Se toman decisiones sobre cómo se estructuran las aplicaciones distribuidas en subsistemas distribuidos, cuáles de estos subsistemas serán componentes configurables y se define la interface de comunicación entre ellos.
- Se identifican las clases activas (threads) y las pasivas (sin control de threads).
- Se decide sobre las características que tendrán los mensajes (sincrónicos y asincrónicos) y qué patrones de comunicación se implementarán.

En esta etapa se diseña la arquitectura del sistema mapeando el modelo de análisis a un entorno operacional dentro del dominio del problema. Se pueden tener consideraciones especiales según las características del mismo, por ejemplo, en sistemas distribuidos, se detallan con qué tipo de mensaje se comunican los componentes entre sí; en el caso de sistemas secuenciales se detallará el diseño considerando los conceptos relacionados a POO (herencia de clases, ocultamiento de información) y en el caso de sistemas concurrentes de tiempo real, se considerarán también conceptos concurrencia de tareas (información real-time, comunicación cliente servidor, acceso distribuido a la información). Relacionado a este último punto, en el libro de Hassan Gomma [RD8] se detallan los criterios a tener en cuenta en la estructuración de tareas concurrentes para decidir si el objeto del modelo de análisis debe ser diseñado como un objeto activo (tarea) o un objeto pasivo en el modelo de diseño. Estos criterios son:

- Tareas de E/S

  Según COMET se debe comenzar por dispositivos de E/S y determinar si debe ser estructurado como tarea de E/S dirigida por eventos, como tarea de E/S periódica o tarea de E/S a demanda.

- Tareas de control

  Se debe analizar cada objeto de control de estado y definirlo como tarea activada a demanda.

- Tareas periódicas

  Se deben analizar las actividades periódicas internas y definirlas como tareas periódicas.

- Otras tareas internas

  Para cada tarea activada por un evento interno, definirlo como tarea controlada a demanda.

Estas guías que propone la metodología, ayudan al resultado final del modelo en esta fase.

El simulador térmico en cuestión aplica el patrón de arquitectura de control centralizado, esto significa que existe un único componente de control (la aplicación STS) que ejecuta la máquina de estados SMP2 y provee el control general del sistema. Este componente recibe eventos desde otro con el que interactúa: la instancia de STSGUI que es la interfaz de usuario del sistema. Un evento de entrada que llega al componente de control implica una transición de estado en la máquina de estados, que resulta en el desencadenamiento de acciones dependientes del estado; como por ejemplo la modificación del tiempo de la simulación y el control de la propagación de la temperatura cuando se detiene y reanuda la simulación.

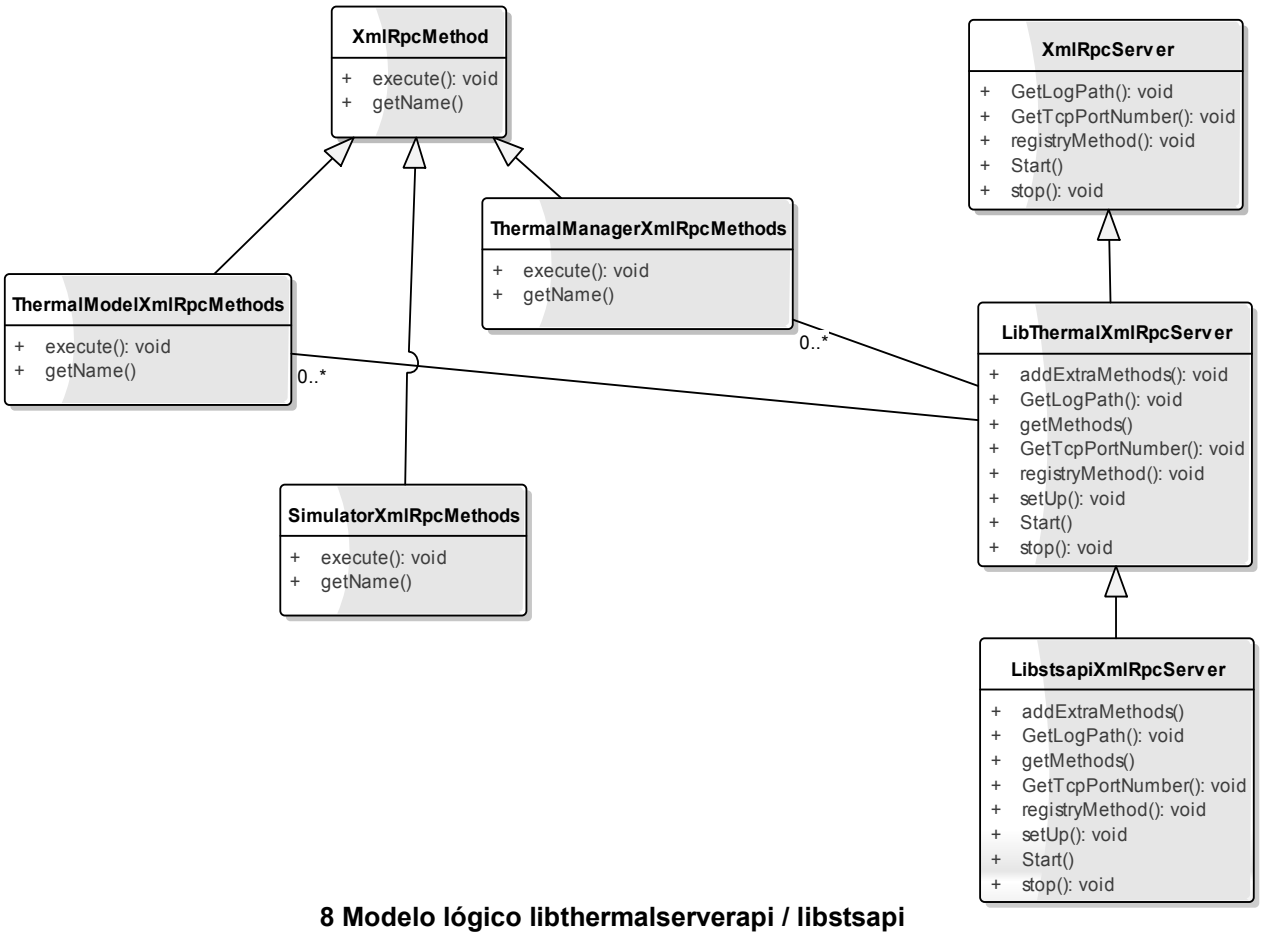
Este simulador es soft real time, esto significa que los requerimientos de tiempo permiten algunas latencias.

Esta aplicación corre en una plataforma cliente/servidor, con comunicación sincrónica provista por la tecnología middleware XML-RPC. Los procedimientos se alocan en el servidor y se invocan por procedimiento remoto. El servidor (proceso STS) recibe un pedido del cliente (proceso STSGUI), invoca el procedimiento apropiado y devuelve una respuesta al cliente.

El patrón arquitectónico cliente/servidor puede representarse por medio de un diagrama de despliegue o por un diagrama de comunicación representando el envío de mensajes sincrónicos. El cliente y el servidor son tareas concurrentes. El cliente está estrechamente unido al servidor, porque envía un mensaje y luego espera una respuesta. Después de recibir el mensaje, el servidor lo procesa, prepara una respuesta y envía la respuesta al cliente. Luego, el cliente reanuda la ejecución.

La desventaja que presenta esta dependencia entre ambas tareas, es que el cliente puede ser retenido indefinidamente si el servidor tiene mucha carga de trabajo. Este desarrollo consta de un único hilo de ejecución para el servidor y otro único hilo para el cliente. Con el avance en conocimiento del dominio, se llegó a la conclusión de que los eventos que ocurren en el ciclo de simulación (EntryPoints) se ejecutan de manera secuencial planificados por la clase Scheduler que es la encargada ejecutar el paso de simulación llamando a los EntryPoints planificados, según el período de tiempo que marca el TimeKeeper. Ver Implementación del estándar: SMP2.

Uno de los incrementos propuesto en una de las iteraciones fue realizar la librería encargada de la comunicación XML-RPC. En particular se implementó un refactoring en registro de métodos al servidor xml-rpc. En el que se realizó la siguiente jerarquía de clases:



8 Modelo lógico libthermalserverapi / libstsapi

Partiendo de la clase XMLRpcServer, se optó por hacer una jerarquía formada por XMLRpcServer <--LibThermalXMLRpcServer <-- LibStsApiXmlRPcServer de manera que los hijos de la librería térmica puedan registrar en el servidor las funciones heredadas de su API.

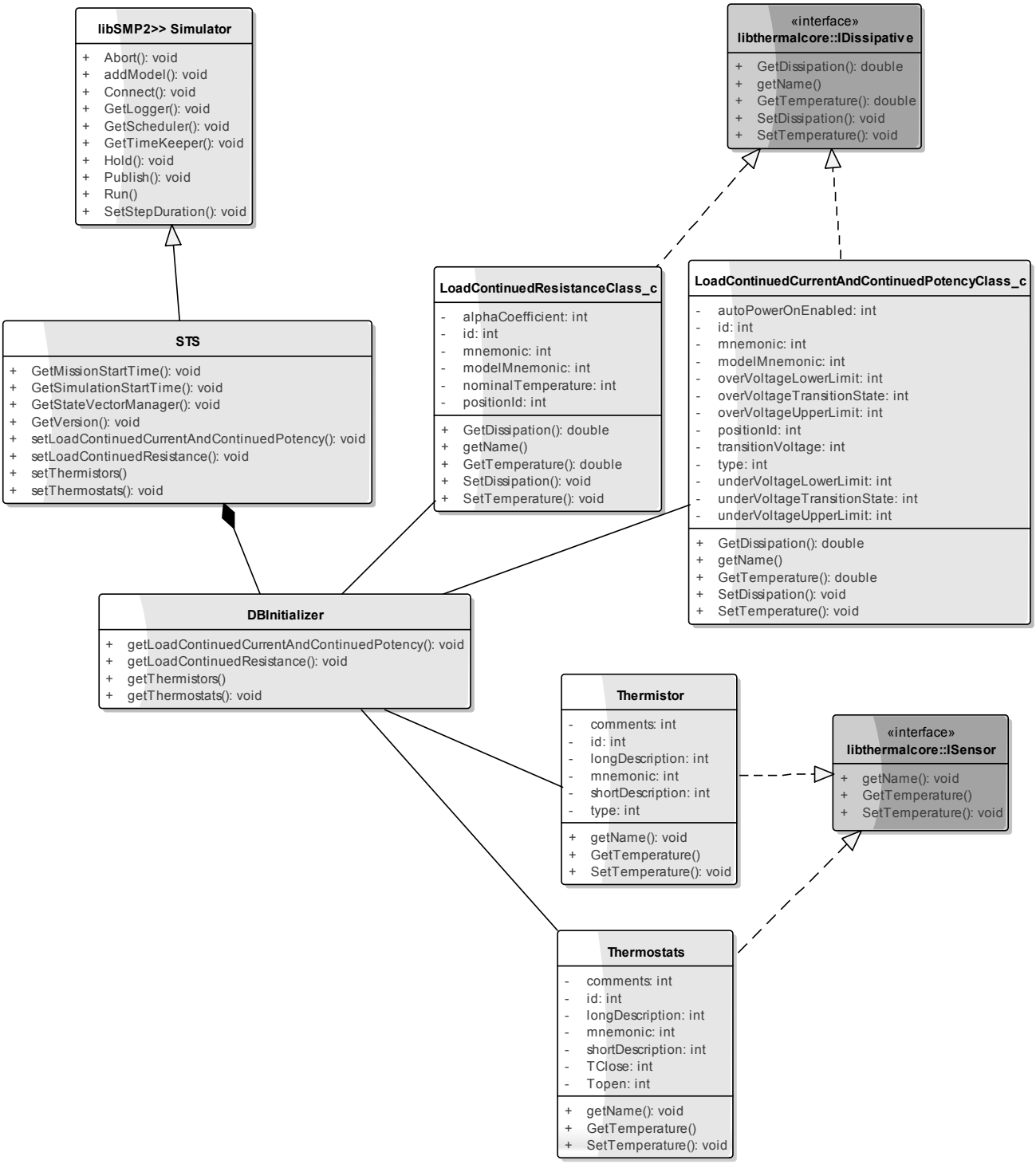
Por otra parte, planear el testing también es una actividad a realizar durante el diseño, en el caso de la librería se propuso un modelo térmico básico que se generaba automáticamente a partir de la clase ObjectModelInitializer_c para realizar pruebas unitarias de la librería térmica y pruebas end to end de la librería libstsmodels.

Construcción Incremental del SW y Testing del Sistema

Estas fases de construcción y testing incrementales, implican codificar el incremento de funcionalidad propuesta para la iteración, integrar y verificar que cumple los requerimientos. En el sistema en cuestión, las primeras etapas estaban relacionadas con la librería térmica y luego con el STS. En el caso del STS los incrementos fueron:

- la creación de un simulador vacío que inicie, pause y detenga una simulación (primera prueba de integración con la librería de simulación)
- configuración del ambiente en el que se simulará la presencia del satélite y tiempos de ejecución (se verifica implementación completa de la interfaz de simulación)

- carga de la base de datos en memoria (integración con librería térmica)



9 Modelo lógico libstsmodels

En este diagrama se presentan las clases que implementan dichas funcionalidades.

Una vez terminado algún incremento propuesto para la libstsmodels y sus respectivos server y cliente, se realizaban pruebas que cargaban disipadores o sensores al simulador y se verificó que se apliquen como nodos del modelo térmico también con sus correspondientes relaciones.

El producto resultado de esta iteración se evaluó hasta que los requerimientos quedaron cubiertos. Las ventajas que se observaron al generar el sistema mediante un prototipo incremental, fueron el

enfoque gradual e iterativo que permitió construir y testear poca funcionalidad por vez, esto facilitó la detección de problemas significativos de manera temprana y en la siguiente iteración se atacó el problema revisando la fase de diseño.

ii. Desarrollo modular a través de librerías

Un desarrollo de este tipo está compuesto por pequeños segmentos de código desacoplados. Estos segmentos atacan diversos aspectos del problema y pueden ser codificados por diferentes equipos de trabajo, cada producto con su propio ciclo de vida y planificación. Los resultados pueden unirse luego.

La modularidad es el mecanismo que permite coordinar este trabajo entre equipos, manejar interdependencias entre partes de los proyectos y ensamblar sistemas complejos de manera confiable. El SW diseñado de esta forma limita el riesgo de acoplamiento progresivo exigiendo que los diferentes componentes del sistema interoperen a través de una API bien definida, pudiendo poseer funcionalidad pública o privada y definiendo las dependencias con otros módulos; en este caso práctico, esos módulos se implementaron como librerías.

Fue de gran importancia contar con un versionamiento de las mismas, para asegurar que estas piezas independientes funcionaran juntas.

Durante la etapa de testing de integración se depuraron las Release Candidates (RC) obteniendo las siguientes versiones:

- Thermalcore tag v.0.3-rc
- libthermalcoreclient tag v.0.3-rc
- libstsclientapi tag v.0.3-rc
- STS tag v0.3-rc
- STSGUI tag v0.3-rc

Una vez superado el Test Procedure (TP) del Apéndice C, se realizaron tags de las Versiones Controladas (VC), de manera que todas las librerías dispongan de una primera versión que, en conjunto, garantizan que cumplen la funcionalidad determinada en el TP. Estas son las versiones finales de las librerías:

- Thermalcore tag v.1.0
- libthermalcoreclient tag v.1.0
- libstsclientapi tag v.1.0
- STS tag v.1.0
- STSGUI tag v.1.0

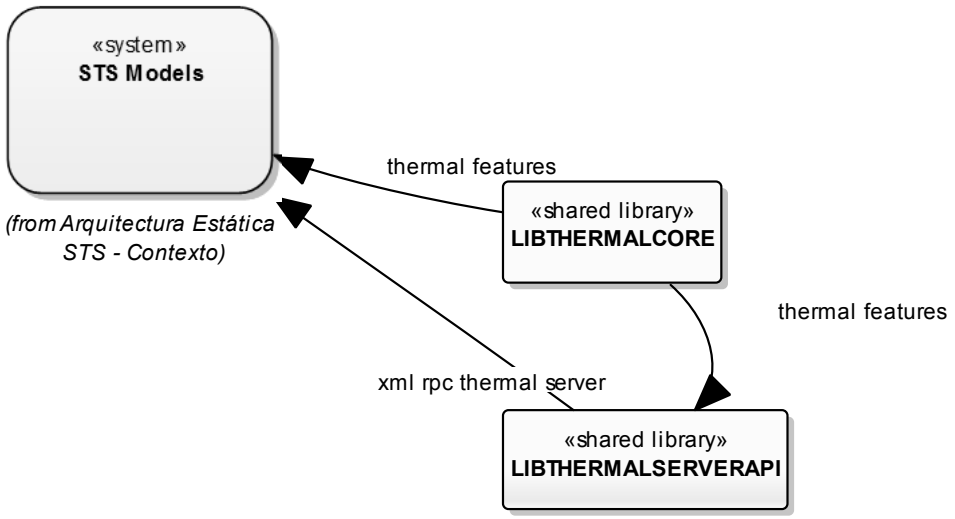
La división de la aplicación en librerías trajo beneficios para la calidad del SW: simplificó el diseño, disminuyó la complejidad de los algoritmos y el tamaño total del programa, ayudó en la reutilización de código, facilitó la depuración, prueba y mantenimiento.

Aunque, dentro de una librería podrían existir malas prácticas de codificación, la modularización se observa en la arquitectura de la aplicación a partir de las dependencias entre las mismas. Si una librería no tiene dependencia con otra, significa que no accede directamente a sus clases. Esto

mantiene la arquitectura limpia previniendo el acoplamiento de segmentos de código no relacionados.

Existen por otra parte, algunas buenas prácticas para el diseño modular, una de ellas es que no deben existir ciclos cuando se crean grafos con la forma en que se interconectan los módulos (A->B->C->A). En caso de que ello ocurriera significa que la separación no fue la correcta, que tal vez dichas funcionalidades no deberían estar separadas o que el modelado de objetos del problema es deficiente.

En el STS, se separaron las responsabilidades tratando de evitar ciclos al incluir las dependencias con librerías propias y con librerías externas. En una primera aproximación a la solución se planteó implementar el servidor de la API junto con el Core del simulador térmico. El primer inconveniente encontrado fue el siguiente ciclo:



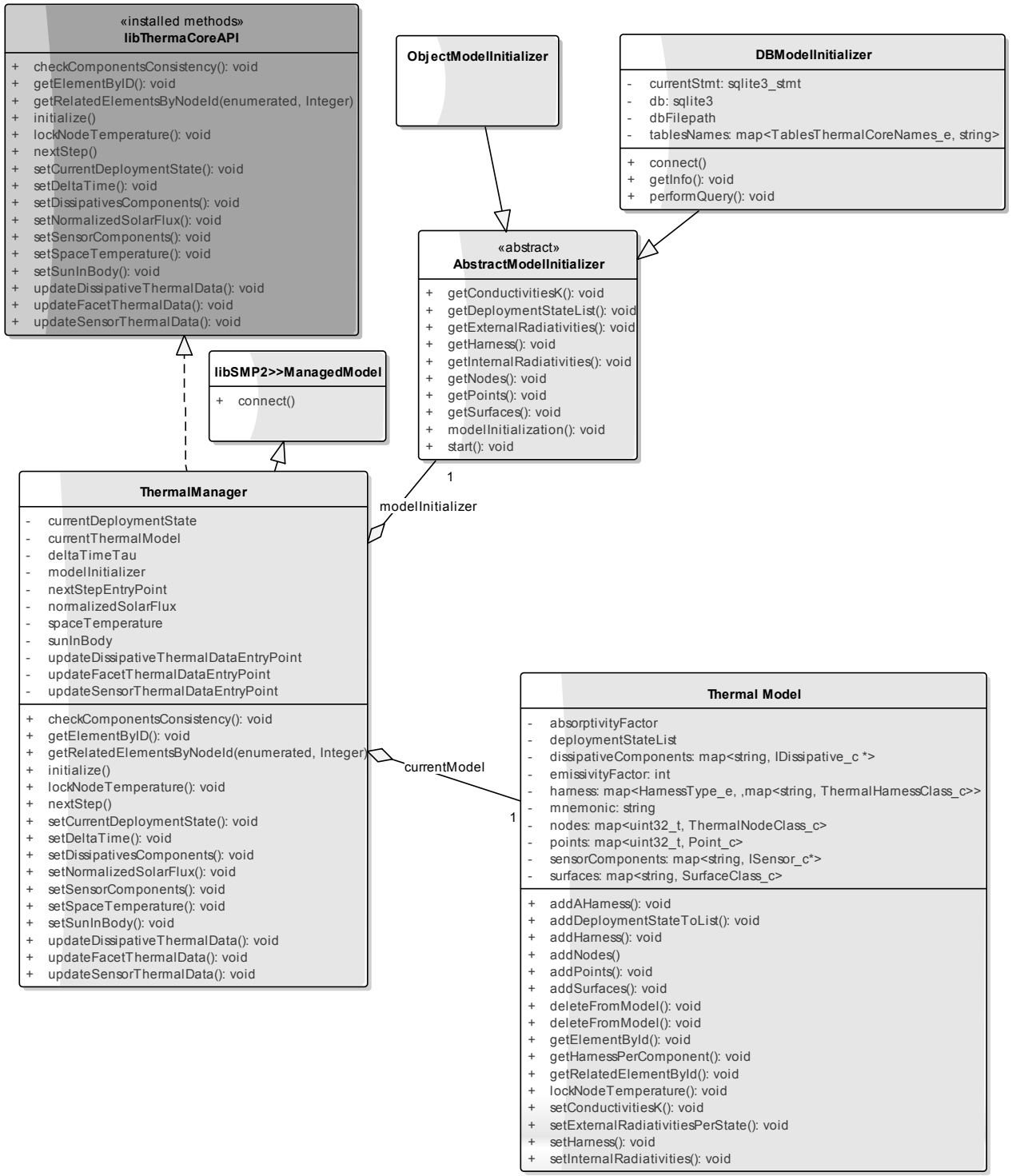
10 Contraejemplo de dependencias en librerías compartidas

Dado que ello no debería ocurrir, se optó por dividir en dos librerías las funciones relacionadas a los modelos y la simulación y, por otra parte la creación del servidor y el registro de métodos en la API.

Por estos motivos es importante en la etapa de diseño entender las subordinaciones entre las partes de la aplicación. De esta manera se crea una infraestructura que permite un desarrollo más robusto.

En la sección “Refactoring de la librería ThermalCore” del documento de Descripción de Diseño del SW (SDD) se detallan algunos cambios realizados para incorporar las directivas de modularización, un ejemplo fue desacoplar las clases térmicas de la estructura de la base de datos, ya que existía en el código de cálculo térmico de otros simuladores, una clase que contenía toda la información para cargar datos a memoria y para calcular la evolución de su temperatura. Se separó esta funcionalidad con diferentes clases:

- ThermalManager_c: configura el modelo térmico y su contexto
- ThermalModel_c: contiene los nodos del modelo térmico y sus relaciones
- ThermalSolver_c: realiza el cálculo térmico de modelo
- AbstractThermalModelInitializer_c: tiene la capacidad de brindarle objetos al modelo, ya sea desde la base de datos o de un modelo base para testing.



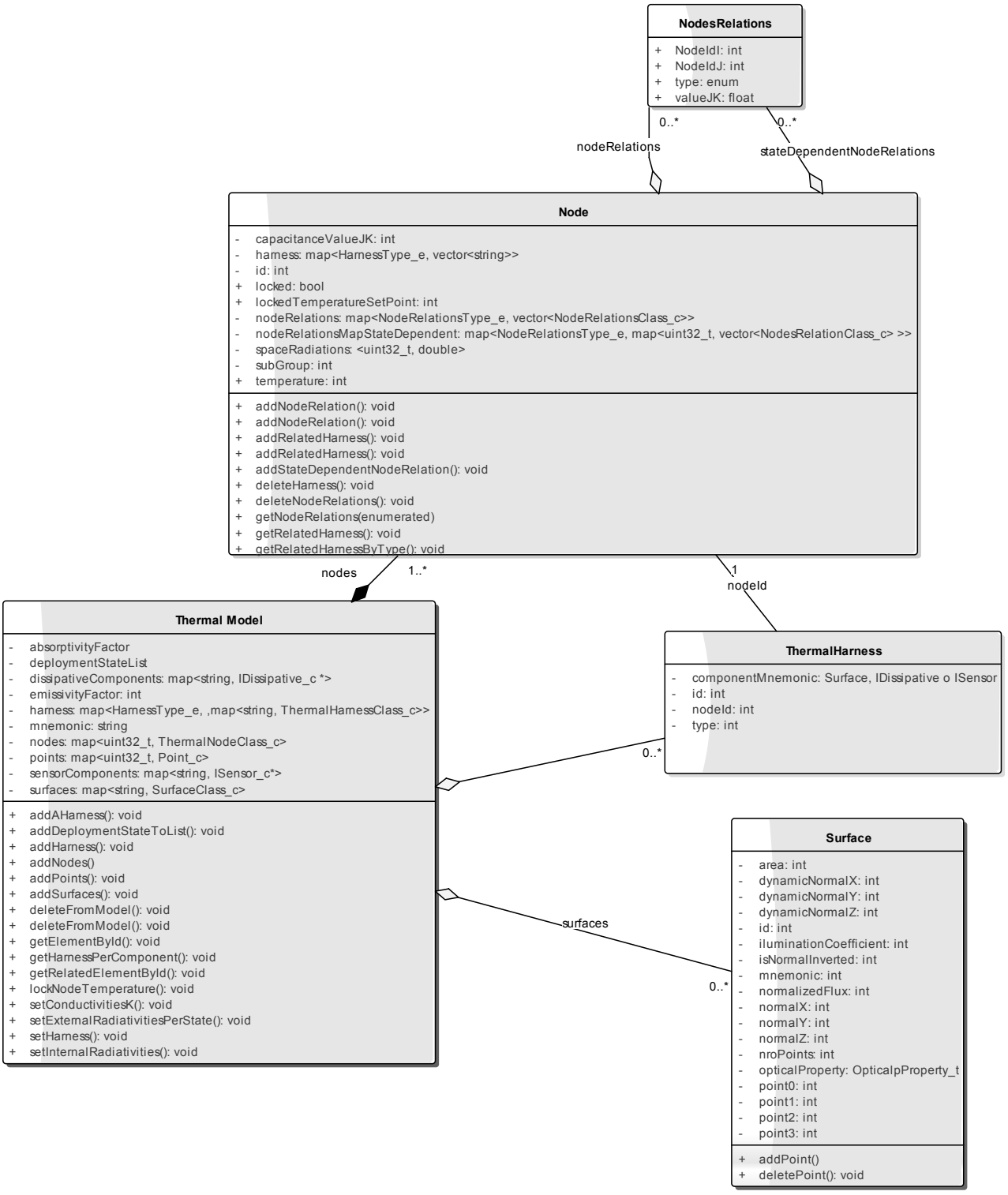
11 Modelo lógico libthermalcore: acceso a datos e implementación SMP2

El desacople de responsabilidades facilitó la lectura del código, ya no se requirió conocimiento previo de la composición de la base de datos térmica y se pasó de una clase Thermal_c de 1274 líneas de código a una ThermalModel_c con 590 líneas de código.

Además la misma clase Thermal_c implementaba SMP2 por lo tanto mezclaba lógica de la simulación y de la planificación en la clase encargada del cálculo. En la librería térmica, el registro

de EntryPoints (parte de la implementación del estándar), quedó a cargo de la clase ThermalManager_c dejando al modelo sólo la responsabilidad de manejar datos.

Las relaciones entre nodos, se separaron entre las que dependen del estado de despliegue y las que no. Por lo tanto son relaciones dependientes las Radiaciones Externas y las no dependientes son las Radiaciones Internas y las de Conductividad. Todas bajo el mismo comportamiento de relación entre nodos como puede verse en el diagrama:



12 Modelo lógico libthermalcore implementación de la malla de nodos

En resumen, la modularidad genera sistemas limpios, con control de las interdependencias entre módulos, no agrega demasiado costo durante el diseño y da a los desarrolladores mayor flexibilidad en el mantenimiento, ya que los mayores beneficios suelen encontrarse en las versiones más avanzadas de las librerías o en su integración.

Librerías estáticas y librerías dinámicas en Linux

Existen dos maneras de compilar librerías, puede ser de manera estática o dinámica. Como ya se mencionó en el inicio de la tesina, el desarrollo del STS fue realizado a partir de librerías dinámicas o también llamadas librerías compartidas. A continuación veremos las diferencias que se proponen en [RD23] para evaluar ambos tipos de librerías:

Las librerías estáticas:

- Son archivos con extensión .a (o .lib en Windows)
- Todo el código relacionado a la librería se encuentra en ese archivo y se linkea directamente al programa en tiempo de compilación.
- El programa que usa la librería, crea una copia del código de la misma y lo hace parte del programa, por lo tanto, el código se repite en cada ejecutable que la utilice.
- En caso de realizar cambios en la librería es necesario saber cuáles son los programas que se verán afectados.
- No utilizan código independiente de la posición en memoria.

Las librerías dinámicas:

- Son archivos con extensión .so. Ej.: libthermalcore.so (o .dll en Windows)
- Todo el código relacionado a la librería se encuentra en ese archivo y se linkea a los programas en tiempo de ejecución, lo que puede acarrear un costo adicional por la carga de las funciones a usar [RD24].
- El programa que hace uso de la librería, referencia sólo el código que manejará. Por lo tanto reduce la cantidad de SW duplicado manteniendo pequeños los binarios.
- En caso de realizar cambios en la librería es necesario reinstalarla. Los programas que la tienen como dependencia la usarán en la siguiente ejecución. Además es sencillo identificar aquellos que serán afectados por el cambio, ya que estos indican cuáles son sus dependencias.
- Utilizan código independiente de la posición en memoria. Lo que le permite al sistema operativo realizar una serie de optimizaciones; en especial porque las librerías consisten de instrucciones que no se modifican y el Sistema Operativo (SO) puede ubicarlas en regiones de memoria Read-only compartida entre procesos. Por lo tanto, si cientos de programas se están ejecutando y cada programa incluye la misma librería, el sistema operativo puede cargar una única copia compartida de las instrucciones de la librería en la memoria física. Esto reduce el uso de memoria y mejora el rendimiento del sistema [RD17].

  Para lograr que el código sea independiente, se realiza un proceso de realocación a través de un mecanismo de indirección Global Offset Table (GOT) y Procedure Linkage Table (PLT). Estas tablas proveen las direcciones de funciones y datos externos que el 'dinamic loader' asigna durante el proceso de carga.

Las librerías creadas por el usuario (como la libthermalcore), son instaladas en el directorio /usr/local/lib a través del comando sudo make install que se detallará más adelante.

Para cargar las librerías compartidas en tiempo de ejecución, se utiliza el comando ldconfig que es el responsable de esta tarea. Crea los links y cache necesarios para encontrar las librerías a usar, indicadas en el archivo ld.so.conf.d, en consola y en las carpetas /lib y /usr/lib. Este comando lee el header del ejecutable que se encuentra en Executable and Linking Format (ELF) y determina qué

librerías cargar y qué versiones, luego realiza el enlace dinámico para arreglar las direcciones de los punteros dentro del ejecutable y las librerías cargadas. Si se modificase el archivo ld.so.conf.d debe volver a ejecutarse el comando ldconfig para hacer un rebuild del ld.so.cache.

En un nivel más alto de programación, se tuvieron en cuenta también algunas buenas prácticas para la conformación de las librerías como por ejemplo:

- Reducir la cantidad de símbolos públicos (ayuda a disminuir la posibilidad de colisión de símbolos que suele ser un gran problema para depurar)
- Basar la versión de release de la librería en la API, es decir, cambiar la versión principal cuando las interfaces se cambian de una manera completamente incompatible, cambiando la versión menor cuando se agrega algo o si los cambios son compatibles hacia atrás, y finalmente aumentando el "patch level" en cada versión.

iii. Implementación de estándar: SMP2

El área espacial de la empresa ha trabajado en asociación con la National Aeronautics and Space Administration (NASA) y la European Space Agency (ESA). Esta relación ha aportado al crecimiento técnico de los desarrollos espaciales, tanto en conocimientos, metodologías y herramientas para implementación de satélites, diseño estructural de los mismos, operación, calidad de procesos, simulación de sus partes como los Paneles Solares, el sistema de Power, AOCS (Attitude and Orbital Control System), etc. En cuanto a la simulación satelital se tomó como herramienta base un estándar empleado y creado en la ESA: la especificación SMP (Simulation Model Portability). La ESA estuvo involucrada en desarrollos de simulaciones espaciales por muchos años ya sea para análisis, ingeniería, testing o entrenamiento. Esta empresa ha estado liderando el desarrollo de la especificación de Portabilidad de Modelos de Simulación en la búsqueda de reducir costos de desarrollo de simuladores. El objetivo de SMP es promover la portabilidad de modelos entre diferentes entornos de simulación y sistemas operativos, promover la reutilización de modelos de simulación, soportar técnicas de Ingeniería de SW modernas como la programación OO y el diseño basado en componentes, lograr simulaciones configurables, por mencionar algunos [RD25]. Este estándar cubre sus objetivos a partir de la reducción de las interacciones de los modelos con el entorno y la estandarización y simplificación de las interfaces usadas por los modelos para que sean fácilmente entendibles por otros desarrolladores. [RD42].

Según el resumen de la evolución del estándar que se realizó en [RD29], SMP ha estado en uso desde 1999. Esta versión se aplicó con éxito a una serie de desarrollos de simuladores dentro de los proyectos de la ESA, pero la falta de soporte para las tecnologías modernas de ingeniería de software dio motivo para iniciar el desarrollo de una nueva versión importante de la norma, el SMP2 en sus versiones 1.0 y 1.2. En 2007 se inició el primer grupo de trabajo en el Comité Europeo de Normalización Espacial (ECSS WG) para promover que la especificación SMP2 se convierta en parte de la serie de normas ECSS. Esto dio como resultado una serie de manuales ECSS liberados como Memorandos Técnicos (TM), que explican los principios fundamentales de SMP2 y que se usaron como capacitación en el desarrollo del STS [RD10] [RD11] [RD25].

SMP está basado en ideas de diseño en base a componentes, arquitectura dirigida por modelos (MDA), UML y XML (eXtensible Markup Language). Uno de sus principios básicos es la separación de aspectos de los modelos de simulación que son específicos de plataforma de los que no. Existen dos plataformas diferentes en la ESA: EuroSim y SimSat. Por ello se buscó proteger los avances de los modelos ante los cambios en las tecnologías que los definen, para que puedan mapearse a alguna de estas dos tecnologías o a nuevas.

Tanto EuroSim como la plataforma SimSat, son usadas para dar soporte a todas las fases de las misiones. Esto significa que se utilizan en diferentes tipos de simuladores que se generan a lo largo del ciclo de vida del satélite. Estos tipos son explicados en detalle en el Memorando Técnico de la ECSS [RD18]:

- Simulador de Concepto del Sistema (SCS)

  Los simuladores de este tipo se desarrollan y validan contra los requerimientos de misión de alto nivel, se encargan de ejecutar modelos matemáticos que no son de tiempo real y de dar apoyo a la ingeniería, permitiendo una rápida evaluación de conceptos de diseño de sistemas.

- Simulador de Ingeniería Funcional (FES)

  Permiten la verificación de elementos críticos del diseño del sistema (como el manejo de datos, algoritmos de AOCS, etc.).

- Banco de Pruebas de Validación Funcional (FVT)

  Permiten realizar un análisis del rendimiento, probar y validar el diseño de un subsistema en el contexto del sistema general con pruebas funcionales.

  Inicialmente, FES y FVT se desarrollan y validan en base a especificaciones preliminares y datos de diseño (obtenidos del PRR –Production Readiness Review- y PDR –Preliminary Design Review-) y luego se van actualizando.

- Simulador de Performance de Mision (MPS)

  Permite la verificación del desempeño general de la misión desde el punto de vista del usuario, incluyendo los modelos de la carga útil (payload).

- Facilidad de Validación de SW (SVF).

  Simula el contexto en el que el satélite va a funcionar: el espacio y la interfaz con segmento terreno. Esto implica control de operaciones, insumos y restricciones ambientales. Contiene un modelo de simulación de funcionalidad y performance del hardware del satélite y su comportamiento dinámico en el espacio.

- Facilidad de Integración y Verificación (AIV)

  Durante la calificación y aceptación, se reemplaza el equipo faltante con esta facilidad y también se simula el ambiente, la posición y la actitud. Esto permite pruebas en tiempo real en las que se tiene en cuenta la respuesta a los telecomandos y a los estímulos ambientales.

  El simulador de AIV comparte muchos puntos en común con el SVF y ambos a menudo comparten modelos e infraestructura comunes. El Simulador de AIV también puede evolucionar para convertirse en el simulador de operaciones.

- Calificación del Sistema de Tierra, Pruebas y Operaciones

  Este SW posee el modelo de comportamiento en tiempo real del segmento espacial, estaciones terrestres e interfaces con el sistema de control de misión para validar el segmento terrestre y capacitar a los operadores. Los modelos de simulación de operaciones comparten características con los modelos utilizados en el SVF y el AIV por lo que deberían derivarse de ellos.

  La verificación técnica y validación del segmento de tierra incluye un chequeo de su compatibilidad con el segmento espacial, utilizando un simulador que incluya por ejemplo, emuladores para elementos de software complejos y en algunos casos componentes reales.

Los diversos casos de simuladores mencionados comparten necesidades y características, por lo que la ECSS recomienda que se construyan como evoluciones, con modelos parcial o totalmente

reutilizados y diferentes requerimientos que deben tenerse en cuenta durante las primeras fases, para que este enfoque tenga éxito.

SimSat y SIMULUS

SimSat es una infraestructura de simulación en tiempo real de uso general; cualquier simulador (incluso no espacial) puede construirse sobre esta plataforma incorporando nuevos modelos. Soporta todos los servicios de simulación estándar, como la programación de eventos en tiempo real, logging, guardado y recuperación de simulaciones, control de la velocidad de simulación, etc. SimSat es parte de SIMULUS, que es un conjunto de simuladores de satélite utilizado en, prácticamente, todas las misiones satelitales de la ESA, tanto durante la preparación de la misión, como en el entrenamiento, las pruebas del segmento terrestre, y el análisis de misiones [RD4]. Esta infraestructura (SIMULUS) está compuesta por el entorno de tiempo de ejecución (SimSat), emuladores de software, un conjunto de modelos genéricos reutilizables y un conjunto de modelos de estación terrena. El objetivo que la ESA persigue con SIMULUS, es utilizar el SW real de plataforma como núcleo de la simulación y las otras partes del satélite representarlas modeladas de manera funcional. Todos los modelos, incluyendo el del procesador en el que corre dicho SW, se ejecutan en el entorno de simulación SimSat [RD12].

Algunos usos no espaciales de SimSat según [RD44] son simulaciones de centros de control para centrales eléctricas, centros logísticos, grandes fábricas o plantas químicas, instalaciones de entrenamiento de tripulación, simulaciones de tráfico, etc.

EuroSim

Otra plataforma de simulación de la ESA es la llamada EuroSim. Es una plataforma de simulación usada para programas espaciales y no espaciales, a través de simulaciones de tiempo real con tecnología Hardware-in-the-loop y reutilización de modelos de SW. La plataforma EuroSim ha avanzado en la implementación de simuladores hard real time cubriendo casos que SMP2 no contempla en sus interfaces. Por ejemplo, en los sistemas hard real time, el mayor problema se encuentra en el ensamble de modelos y sus interacciones. Es decir, si los modelos se ejecutan en simultáneo y no se incluye control de acceso a los datos, es posible que utilicen datos incorrectos o inconsistentes. Además, el hecho de introducir control de acceso a los datos, genera el problema de los tiempos de ejecución no deterministas (en el caso de que un acceso se bloquee). Por otra parte, la interfaz de planificación de tareas de SMP2, no permite especificar la ejecución en varias CPU. Todas estas cuestiones, como se explica en el paper [RD5], EuroSim las tiene en cuenta e incluye interrupciones, señales, semáforos, mutexes, prioridades en tareas, y otras técnicas de programación concurrente, en pos de alcanzar requerimientos de tiempo estrictos.

El diseño de EuroSim se basa en el principio de que cada simulador de mayor alcance puede descomponerse fácilmente en simuladores más pequeños. EuroSim ayuda a reducir los costos asociados con esta actividad.

La ESA aplicó EuroSim en el proyecto ATV (Vehículo de Transporte Autónomo 2008), en el ERA (European Robotic Arm 1996), en el desarrollo y la verificación de los satélites Herschel, Planck y Gaia de la Agencia Espacial Europea, también en el ámbito de defensa, como plataforma de simulador de vuelo, así como entorno de desarrollo y prueba para la formación integrada de buques múltiples a bordo del F-35 Lightning II., también en el proyecto Fighter 4-Ship (F4S) que es una instalación que puede simular las operaciones tácticas colectivas de hasta cuatro aviones de combate, entre otros [RD43].

En el marco de la interoperabilidad y reutilización de modelos existen otros estándares que complementan el trabajo del SMP2 de manera diferente en particular se analizaron los casos de HLA y Modélica.

Modelica es un lenguaje de alto nivel para desarrollar modelos de sistemas basados en física desde el diseño arquitectónico hasta el nivel de componentes. Es un estándar abierto (no propietario) para modelado de sistemas, al igual que SMP2 es orientado a objetos y sirve para

modelos espaciales y no espaciales. Los modelos generados bajo este estándar son portables ya que se basan en una interface funcional maquetada (FMI – Functional Mockup Interface) que es un estándar que permite el intercambio de modelos compilados entre herramientas y permite también la simulación distribuida de los diferentes subsistemas de un mismo dominio. Posee un fuerte apoyo de la industria automotriz y su interfaz es soportada por más de 95 herramientas de Ingeniería Asistida por computadoras [RD9].

En el SESP2017, foro en el que representantes de las Agencias Espaciales y de la Industria pueden presentar y debatir las actuales tendencias y necesidades del estado actual de esta área, se trataron estos estándares como solución a la conectividad entre la simulación y las herramientas de ingeniería y como vínculo entre los requerimientos y el diseño basado en modelos.

La ESA considera que la simulación distribuida es importante para el beneficio de proyectos espaciales ya que los mismos se desarrollan en compañías distribuidas en todo el mundo y desean reducir la duplicación de trabajo; por lo tanto la interoperabilidad de las simulaciones optimizaría el uso de los recursos de equipos de trabajo distribuidos. Por este motivo la Agencia, seleccionó High Level Arquitecture (HLA estándar IEEE 1516) para su evaluación [RD45]. HLA es un estándar esponsoreado por el Departamento de Defensa de EEUU (DoD) provee las bases estructurales para interoperabilidad de simuladores a partir de un conjunto de reglas a las que deben adherirse todas las aplicaciones compatibles, un modelo común (Object Model Template OMT) para compartir datos y documentar datos compartidos, y una interfaz de programa de aplicación (API) para un sistema de software RTI (Runtime Infrastructure) que integra los simuladores individuales. El RTI proporciona, entre otras cosas, un protocolo de comunicaciones y de intercambio de datos, mecanismos para gestionar el tiempo, y un método de sincronización de los simuladores interactivos. El esfuerzo requerido para integrar simuladores compatibles con HLA en sistemas de simulación más grandes es mucho menor de lo que sería necesario para integrar simuladores similares que no fueron diseñados de acuerdo con una arquitectura e interfaz comunes [RD46].

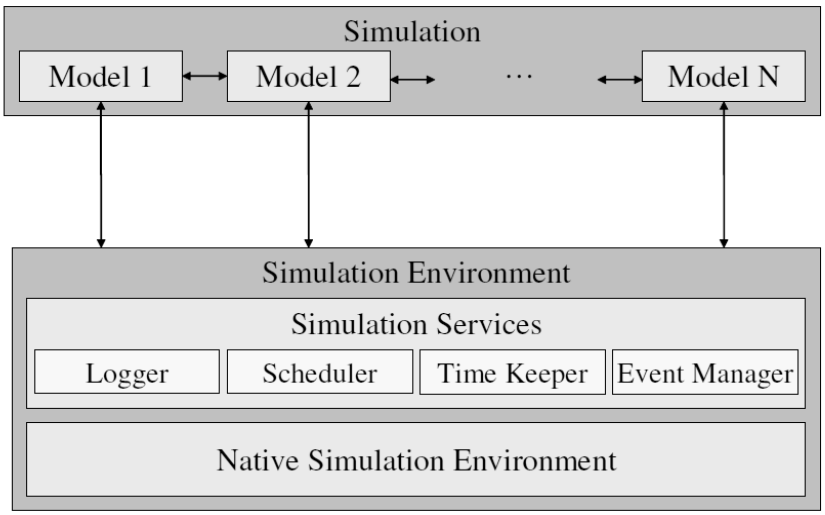
HLA busca interoperabilidad entre simuladores y SMP2 busca reutilización de componentes entre diferentes simuladores. Asi SMP2 reduce el costo, el tiempo de desarrollo de sistemas de simulación y aumenta sus capacidades facilitando el reúso y la operación entre sus componentes, pudiéndose usar un mismo componente en simuladores con diferentes propósitos.

Los elementos que forman parte de SMP2 son:

- **SMP Meta-modelo:** define componentes, interfaces, jerarquías, relaciones, herencia y tipos.
- **Modelo de Componentes Abstractos:** define la funcionalidad básica que el modelo de componente debe tener y el patrón Factory para instanciarlos.
- **Servicios de simulación:** serán mencionados más adelante. Son necesarios para el entorno de simulación
- **Mapeo del meta modelo:** a implementaciones específicas en lenguajes como C++, Java, C#, etc.

Como indica el último de los elementos, existe una especificación de cómo mapear la teoría del estándar a interfaces en un lenguaje de programación. De esta manera fue creada la librería que usa el STS como dependencia, la cual es la encargada de proveer los servicios de simulación y los modelos de componentes abstractos. Esta librería libsmp2 está desarrollada en C++. Ofrece la posibilidad de reutilizar el código en los diferentes simuladores que se lleven a cabo, con la garantía de que ya fue testeado y corregido en desarrollos previos al STS. Al extender clases del negocio de la libsmp2, las clases de la librería libstsmodels, heredan la funcionalidad del estándar, de esta forma se cuenta con la implementación de problemas comunes de modelado. Como consecuencia de esta decisión de diseño se obtienen modelos portables que pueden reutilizarse en otras soluciones de simulación que implementen SMP2.

El estándar SMP2 propone una arquitectura modular. La misma cubre dos tipos de componentes: la Simulación con instancias de Modelos que proveen el comportamiento específico a la aplicación, y el Ambiente de Simulación que provee los servicios de simulación. Esta arquitectura se encuentra implementada en la librería libsmp2 de la siguiente manera:



13 Servicios de Simulación – Imagen recuperada de RD25

Los servicios que propone la arquitectura implementan la interface Smp::IService:

- **Logger:** Permite el logging de mensajes de diferentes tipo, este servicio se implementó usando google-glog.

- **Scheduler**: El scheduler es el encargado de llamar a los entry points basados en eventos temporales o cíclicos, depende fuertemente del tiempo marcado por el Time Keeper. Implementa Smp::Services::IScheduler. El método DoStep() ejecuta un paso de simulación, llamando a los EntryPoints según hayan sido programados.

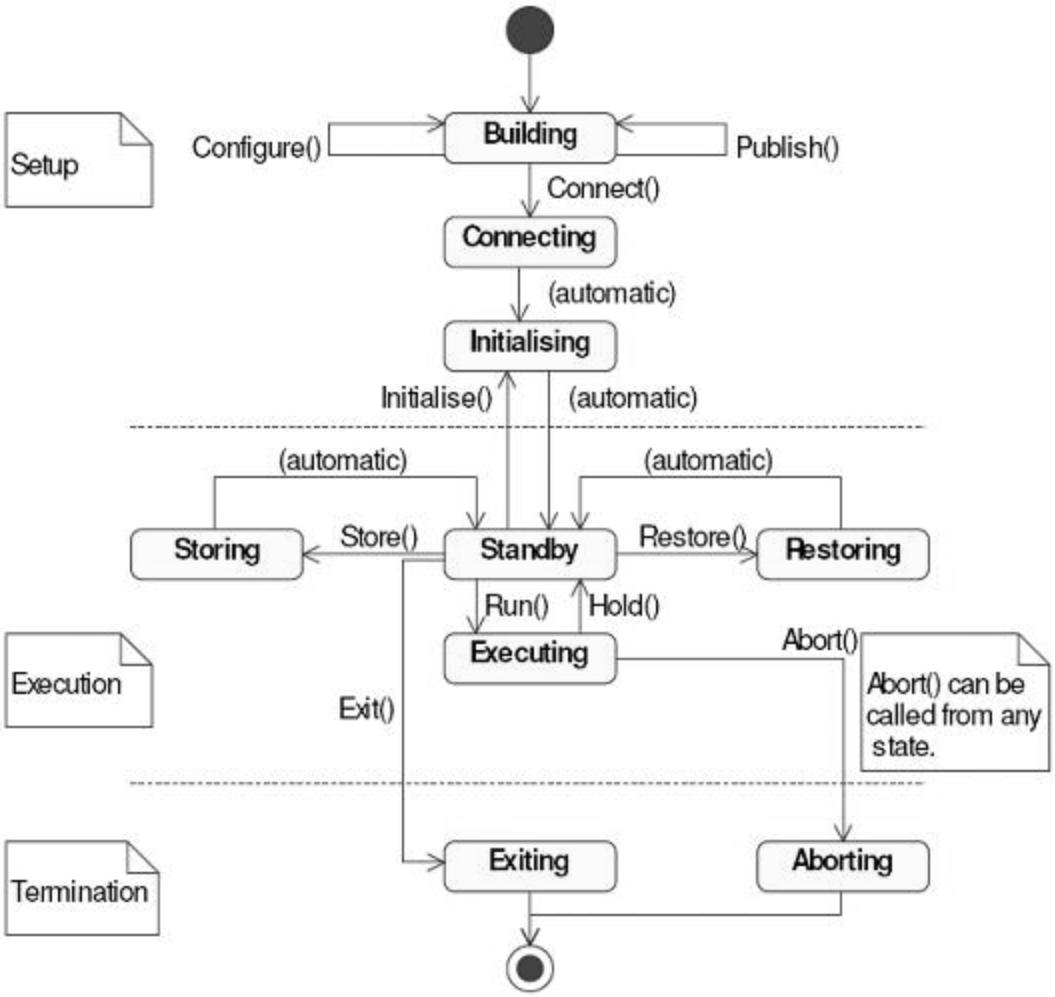
- **TimeKeeper**: Este servicio provee cuatro tipos de tiempos del SMP2. Estos son: tiempo de simulación relativo, tiempo de época absoluto, tiempo relativo de misión y el tiempo relacionado al reloj de la computadora. Implementa el servicio Smp::Services::ITimeKeeper.

- **EventManager**: Este servicio ofrece un mecanismo de notificación global. Los componentes pueden registrar entry points a un evento global a través de manejadores de eventos, se puede realizar broadcast de eventos y definir nuevos tipos. Para que una función pueda ser considerada entry point y de esta manera pueda ser agregada en el Scheduler o en el Event Manager, debe ser una instancia de IEntryPoint y no debe poseer parámetros ni valor de retorno

Una instancia de ISimulator lanza un nuevo thread donde correrá la simulación. ISimulator provee las interfaces de control de la simulación, como son Run(), Hold(), etc., y además del acceso a los modelos y servicios mencionados previamente. La simulación avanza de a ciclos de simulación que el TimeKeeper controla y en cada ciclo se ejecutan una serie de EntryPoints planificados por el Scheduler. Un EntryPoint es una tarea que se registra en el ciclo de simulación. Al agregarse en el Scheduler se indica con qué periodicidad éste deberá ejecutarla y en qué orden.

Para generar un modelo para la simulación es necesario que una clase implemente la interface IManagedModel para dar acceso de sus atributos al manejador de modelos entre los que pueden encontrarse los EntryPoints.

Siguiendo esta arquitectura se creó el STS_c que hereda de la clase ISimulator y es, para la aplicación, el ambiente de simulación. El mismo implementa los métodos Configure(), Publish() y Connect(). Cada uno encargado de hacer broadcast de estos métodos a los modelos que componen el simulador. Estos métodos son indispensables para la inicialización del ambiente, obteniendo como resultado final, la simulación en estado StandBy según el siguiente diagrama de estados:



14 Diagrama de estado del entorno de simulación – Imagen recuperada de RD25

- **Método Publish():** Este método informa al entorno de simulación que se crearon nuevas instancias de modelos y que pueden ser publicadas. Con este método el entorno de simulación le indica a cada modelo que se publique. Es llamado por la aplicación luego de crear todos los modelos.

- **Método Configure():** Informa al ambiente de simulación que todas las instancias de modelo fueron configuradas y que puede ejecutarse su propia configuración. Con este método el entorno de simulación le indica a cada modelo que se configure. Este método es llamado por la aplicación luego de publicar todos los modelos.

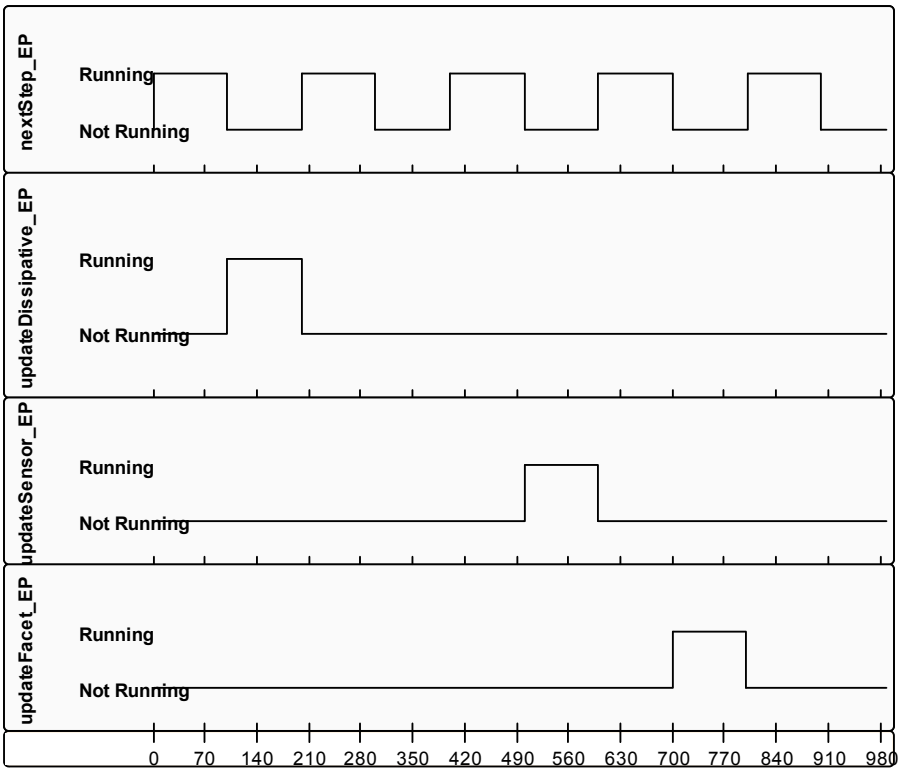
- **Método Connect():** Informa al ambiente de simulación que se creó y configuró la jerarquía de modelos y que pueden conectarse al simulador. El entorno le indica a cada modelo que se conecte con el simulador. Se llama desde la aplicación luego de publicar y configurar todos los modelos.

El STS y la ThermalCore son aplicaciones con un sólo hilo de ejecución y no presentan situaciones en que deban compartir recursos debido a que se espera que pueda ser utilizado solo por un cliente a la vez, que realice las llamadas a procedimiento.

Tanto la librería térmica como el simulador que heredan de clases de la libsmp2, poseen acceso al Scheduler. El mismo le asigna un momento de ejecución para cada procedimiento (entry points) de la simulación.

Los entry points actúan como tareas que se ejecutan periódicamente al registrarlas en el Scheduler. La clase STS_c posee el entry point que calcula la velocidad real de simulación que se está alcanzando en base a las configuraciones y las capacidades del HW. La clase ThermalManager_c posee los entry points que actualizan la temperatura de los sensores, disipadores, superficies y uno que ejecuta el siguiente paso del cálculo térmico acumulando la potencia generada y restando la disipada.

Ambas clases son las encargadas de registrar sus tareas en el Scheduler, por lo que el ciclo de ejecución queda de la siguiente manera:



15 Ciclo de simulación: ejecución de Entry Points

Hacia este último tiempo se creó un nuevo grupo de trabajo en el ECSS para tratar mejoras al SMP2: el ECSS SMP. Se espera que este año se libere una versión para revisión y prueba dentro del comité. Según el material publicado luego del SESP 2017 [RD29], que explica la evolución de SMP2 a ECSS SMP, se puede mencionar que esta nueva edición del estándar sólo estará disponible para C++, algunas cuestiones presentes en el SMP2, como el uso de servicios de simulación opcionales, se eliminaron. Se agregaron conceptos de inicialización a partir de varios archivos de configuración, modelado de errores, entre otros.

iv. Librerías legacy

Fue importante para este desarrollo, la inducción en las diversas tecnologías, herramientas, implementación de estándares, modelos portables y creación de librerías para los simuladores anteriores. Entre ellas se encontraban las librerías ya implementadas para cubrir funcionalidades recurrentes en el dominio de simulación. Estas sirvieron como experiencia en el desarrollo modular a través de librerías compartidas y dieron lugar al desarrollo de la actual tesina cuyo objetivo es separar la funcionalidad térmica en una librería dedicada para facilitar su uso.

Por este motivo el STS cuenta con algunas librerías heredadas como la ya mencionada libsmp2 que actualmente se encuentra corriendo en producción siendo parte de los simuladores de satélite que se utilizan para depurar maniobras de operación antes de realizarlas en vuelo en los Arsat. El hecho de que una librería sea heredada significa que es un paquete de SW realizado hace tiempo que aún están en uso, funcionan bien y que incorporan diseños o técnicas que no son de uso común debido a su antigüedad.

Usualmente ocurre con las librerías legacy que:

- Se desea actualizar pero no se puede o está en medio de una actualización de cambios significativos en el diseño.
- Se pierde el mantenimiento del SW. Es evidente que si la aplicación no se mantiene con regularidad, no seguirá el ritmo de los principales cambios en el mundo del software. Esto podría deberse a una elección deliberada basada en prioridades de negocios o restricciones presupuestarias.
- La falta de Testing se transforma en la causa de que el código sea heredado, ya que se torna difícil una actualización dado que podría dejar de funcionar algo.
- El código fuente ya no está disponible, lo que implica que el sistema sólo puede ser agregado (no modificado).

El código heredado sin embargo, no siempre es nocivo, es una relación entre el costo de modificarlo y los beneficios que ofrece (incluyendo bajo mantenimiento a futuro) [RD26]. En el STS, como se mencionó previamente, se utilizó la librería libsmp2. No obstante fue una decisión de diseño no utilizar la librería libcma que veremos en más detalle a continuación.

Lib cma

La libcma tuvo su primera release en el año 2012 e implementa el mecanismo Component Model Adapter (CMA) que es el que se encarga de gestionar los recursos necesarios a nivel de telemetrías y comandos de una determinada representación de un componente, es decir su modelo.

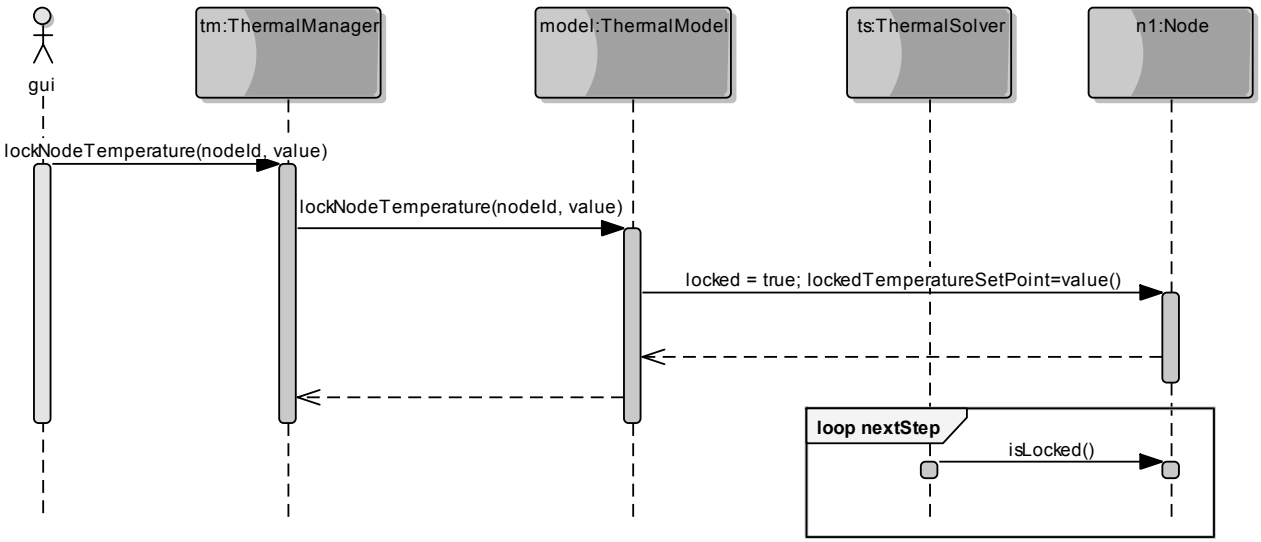
En particular para el STS era necesario usar las Variables de Modelo que son aquellas que lo caracterizan y se disponibilizan en el contexto del CMA para permitir su consulta y sobreescritura. De esta forma se puede acceder a la información de manera aislada para su depuración a través de pruebas y para desarrollos incrementales, independientemente del estado de avance de otros modelos de subsistemas.

Para utilizar dicha función es necesario heredar de la clase ComponentModelAdapter_c. que incluye también el manejo de telecomandos y telemetría. Lo cual no aportaba a la limpieza del código y era semánticamente incorrecto en el caso de un modelo térmico que no tiene lógica que haga uso de esas cuestiones.

Se evaluaron entonces las siguientes opciones:

- costos de heredar funcionalidad semánticamente incorrecta contra los beneficios de disponer de las variables de modelo en el contexto de CMA
- costos de hacer una librería más simple extrayendo esa funcionalidad para independizarla de la libcma y que esta a su vez ésta la reutilice
- implementar la funcionalidad de sobrescribir valores en el modelo térmico

La segunda opción era una buena inversión a largo plazo que implicaba el costo de integración de la nueva librería de manejo de variables de modelo con una libcma que solo se encargue de telemetría y comandos; y el costo del testing necesario para que se instale en los proyectos que utilizan la libcma actualmente. Por otro lado la última opción fue la que mejor se adaptó al problema debido a que el diseño estaba preparado para que en el cálculo de temperatura realizado por la clase ThermalSolver_c, evalúe a cada nodo por su estado (con la temperatura lockeada y sobrescrita o no) para tomar la acción correcta. Fue la solución más económica y sencilla quedando la comunicación de los objetos de este modo:



16 Diagrama de secuencia: Caso de Uso Lock Temperature

El modelo térmico es el encargado de identificar el nodo que se desea lockear, le cambia su estado a locked y le asigna un valor de temperatura fijo. Luego, durante la iteración en la que se ejecutan los EntryPoints de cálculo térmico, es el Solver quien toma decisiones diferentes según el estado de cada nodo.

v. Librerías y testing

El testing de este proyecto fue haciéndose en etapas de manera bottom-up, es decir, primero se probó el core del funcionamiento térmico, luego la capa de comunicación xml-rpc, la carga de datos al simulador y finalmente las pruebas transversales que abarcaban el uso de la GUI, la incorporación de datos al modelo térmico y el cálculo térmico en sí, teniendo en cuenta la configuración del entorno del satélite.

La principal ventaja de la implementación del sistema a través de librerías, fue que esta evolución bottom-up fue natural. Los tests sirvieron como retroalimentación del diseño y la codificación ya que para lograr una completa batería de pruebas, las responsabilidades debían estar bien asignadas y las clases desacopladas.

Estos test facilitan la modificación de código y su mantenimiento, y permite que los errores se encuentren en etapas tempranas del desarrollo.

Los test unitarios además garantizaban el correcto funcionamiento de las librerías de forma aislada, por lo que al conectarlas entre sí, fue sencillo depurar los errores. Tal fue el caso de los test de la libthermalserverapi y la libthermalclientapi cuyo funcionamiento depende fuertemente de la Libthermalcore.

Los test de la Libthermalcore se realizaron en CppUnit verificando el funcionamiento de las clases ThermalManager_c, ThermalModel_c, ThermalSolver_c, Node_c, ObjectModelInitializer_c (carga un modelo térmico mock) y DBModelInitializer_c (carga un modelo térmico desde la base de datos térmica).

Por otro lado los tests de la librería libstsmodels se hicieron en scripts Python. Como se detallará más adelante, las interfaces de las librerías implementadas en C++, se wrappearon a Python para poder invocarlas desde una GUI implementada también en ese lenguaje. Estos test verifican que se puede consumir datos desde la librería térmica y verifican también, la evolución de la temperatura de los nodos. Para completar el testing, en la librería libstsclientapi se comprueban los métodos de la interfaz ISimulator (Run, GetSimulationState, GetSpeedFactorCoefficient, GetSimulationTime, etc.). Existe un framework de testing unitario para Python: Pyunit que no ha sido utilizado en los scripts de testing del STS, por el contrario se realizaron los chequeos a partir de sentencias assert. Es recomendable el uso de este framework para que sean más legibles los test y para reducir el esfuerzo de escribirlos. Pyunit propone una manera sencilla de reutilizar scripts previos (old-style test code) para adoptar el framework de manera menos costosa [RD47].

No obstante no todos los errores se detectan mediante test unitarios. La experiencia con el STS fue, que una vez implementado en su totalidad con su interfaz gráfica casi completa, se descubrió que con grandes volúmenes de datos en el modelo y con el correr del tiempo, el simulador se volvía más lento. Ante este diagnóstico se utilizó la herramienta de análisis Valgrind y se detectaron memory leaks.

En cuanto al testing funcional se tuvieron en cuenta casos base en los que se verificaba el comportamiento de la interfaz gráfica y del simulador en su conjunto. Los Casos de Prueba pueden observarse en el Apéndice C donde se proponen los ejemplos de uso en base a un modelo térmico acotado.

Capítulo 3: Herramientas de desarrollo

i. C++ y Testing Unitario

Lenguaje de programación C++11

El proyecto de esta tesina fue codificado en C++ por requerimiento. Este es un lenguaje orientado a objetos y multithread. Se utilizó en la versión 11 del estándar. Es la cuarta revisión del lenguaje y tomó el nombre del año en el que la Organización Internacional para la Estandarización (ISO) lo aprobó. Su nombre oficial es ISO/IEC 14882:2011.

C++11 es soportado por compiladores modernos como es el caso de g++, utilizado en el proyecto del STS. En el artículo [RD57] se hace referencia a una lista de compiladores y las funcionalidades que soporta este estándar. En algunos casos hay que habilitar la compatibilidad en el compilador, por ejemplo en GCC es necesario compilar con la opción -std=c++0x.

Existe una versión más nueva del estándar: C++14. La cual es de menor extensión y menor cantidad de cambios respecto de C++11. Se está desarrollando también el estándar C++17 que se espera sea lanzado a finales del 2017.

El estándar utilizado posee un conjunto de nuevas funcionalidades que se listan en [RD56] y en [RD57] de las cuales resaltan la inclusión del paradigma de promesas/A+ y la inclusión de librerías para programas de múltiples hilos. A continuación se detallan algunas con las que se tuvo experiencia:

- Enumerativos fuertemente tipados

```
enum class DiasHabiles { Lunes, Martes, Miércoles, Jueves, Viernes};

DiasHabiles dia = DiasHabiles::Martes;
```

- Funciones lamda

Es una función con cuerpo pero sin nombre y se declara de la siguiente manera:

```
[](ListaDeParametros) TipoQueSeRetorna{ CuerpoDeLaFuncion}(ParametrosReales);

double dUpperPart = [](double dX, double dY)double{ return dX*dX +dY*dY;}
```

- Deducción de tipos automática

El compilador puede determinar el tipo de la variable a partir de su inicialización, esto es particularmente útil en el uso de iteradores.

```
int x = 3;

auto x = 3;
```

- Destructor default

Obliga al compilador a generar un destructor para la clase.

```
~ UnaClase() = default;
```

La experiencia de aprender C++ a partir de un conocimiento previo de C implicó profundizar en la sintaxis y en las nuevas posibilidades que este lenguaje ofrecía. La diferencia más evidente entre ambos es que en C++ la implementación es orientada a objetos. En base a esto es importante hacer un resumen de algunos detalles técnicos que se fueron aprendiendo a medida que se progresaba; los mismos facilitaron la codificación dándole un estilo C++. Se seleccionaron desde [RD58] un subconjunto de aspectos a tener en cuenta:

- Las palabras clave *new* y *delete* se usan para alocar y liberar memoria, es mejor utilizarlas en lugar de malloc y free. El delete llama al destructor y libera memoria.
- Los miembros de una clase se inicializan en forma de lista y lo mismo con miembros de la clase base:

```
class Foo {

    int _value;

public:

    Foo(int value=0) : _value(value) { };

};
```

- Cada clase posee sólo un método *destructor*, sin argumentos y sin valor de retorno, el cual no será explícitamente llamado, sino que será utilizado por el delete.
- Los *namespaces* permiten agrupar clases bajo un mismo alcance. De esta manera dos clases con igual nombre pueden convivir en namespaces diferentes y pueden ser llamadas de la siguiente manera:

```
PrimerNamespace::UnaClase

SegundoNamespace::UnaClase
```

- Existe la posibilidad de especificar *valores por defecto para los parámetros* de los métodos, estos deben ubicarse preferentemente hacia el final de la lista

```
float foo( float a, float b, float c=2 )
```

- Las *referencias* permiten declarar alias de otras variables y tienen el mismo tiempo de vida que la variable. Son particularmente útiles en los argumentos de las funciones.
- Existe la posibilidad de especificar que una función es escrita para un tipo de datos genérico a través de *templates*.
- Una clase es *abstracta*, si tiene al menos un método abstracto, es decir al menos un método con el modificador virtual.
- Es importante en la herencia, tener presente el *control de acceso a los atributos*. Por defecto todos los atributos de una clase son privados (excepto que se declaren public o protected). Además se tendrá acceso a los atributos heredados según se especifique en la declaración de la clase base. En el siguiente ejemplo las partes públicas y protegidas de Foo, se mostrarán como protegidas para la clase hija:

```
class Bar_protegida : protected Foo  { /* ... */ };
```

- La palabra clave *friend* se utiliza para dar acceso privilegiado a ciertas clases. Esto es útil en el caso anterior.

Librería de Testing Unitario: CppUnit

CppUnit es un framework para realizar pruebas unitarias en C++.Este framework consta de una jerarquía conformada por TestSuite, y contiene TestCases que prueban funciones específicas de cada clase. Es decir que para crear nuevos casos de prueba es necesario que los mismos hereden de la clase TestCase provista por el framework o bien de la clase TestFixture que provee al test, el método setUp() encargado de la inicialización de las precondiciones y el método tearDown() con el que se restauran las estructuras de datos al finalizar la ejecución del test. A través de sentencias assert() se realizan las verificaciones pertinentes.

En la implementación de las pruebas unitarias de la librería térmica se utilizaron macros propuestas por el mismo framework para facilitar la definición de test suites. Estas macros pueden encontrarse en <cppunit/extensions/HelperMacros.h> y realmente son de mucha utilidad para reducir las líneas de código de creación de los test y reducir también la curva de aprendizaje del framework.

El siguiente es un ejemplo de creación de TestSuite:

ThermalManagerClass_Test.h

```
14  #include <cppunit/extensions/HelperMacros.h>
15  #include "ThermalManagerClass_c.h"
16
17  class ThermalManagerClass_Test : public CppUnit::TestFixture
18  {
19
20      CPPUNIT_TEST_SUITE( ThermalManagerClass_Test );
21      CPPUNIT_TEST( ThermalManagerClass_Test::CheckCreation );
22      CPPUNIT_TEST( ThermalManagerClass_Test::initialize );
23      CPPUNIT_TEST( ThermalManagerClass_Test::nextStep );
24      CPPUNIT_TEST( ThermalManagerClass_Test::resetModelForTesting );
25      CPPUNIT_TEST_SUITE_END();
26      private:
27
28          INVAP::THERMALCORE::ThermalManagerClass_c * manager;
29
30      public:
31
32          void setUp(void);
33
34          void tearDown(void);
35
36          void CheckCreation(void);
37
38          void initialize(void);
39
40          void nextStep(void);
41
42          void resetModelForTesting(void);
43  };
```

C++ source file length : 1109 lines : 46

17 TestSuite de ThermalManagerClass

Aquí se puede observar el uso de dos macros: CPPUNIT_TEST_SUITE que inicia la declaración de un nuevo test suite y CPPUNIT_TEST que agrega un método a dicho conjunto de pruebas en el header de la clase.

ThermalManagerClass_Test.cpp

```
 9  #include "ThermalManagerClass_Test.h"
10  #include "ThermalManagerClass_c.h"
11  #include "ThermalSolverClass_c.h"
12  #include "Mocks.h"
13  #include "ObjectModelInitializerClass_c.h"
14  #include "stdlib.h"
15
16  #define EPSILON 0.005
17
18  CPPUNIT_TEST_SUITE_REGISTRATION( ThermalManagerClass_Test );
19
20  void ThermalManagerClass_Test::setUp(void)
21  {
22      manager = new INVAP::THERMALCORE::ThermalManagerClass_c();
23      manager->initialize();
24      CPPUNIT_ASSERT(manager->getCurrentThermalModel().getNodes().size() > 0);
25  }
26
27  void ThermalManagerClass_Test::tearDown(void)
28  {
29      delete manager;
30      INVAP::THERMALCORE::ThermalSolverClass_c::resetTables();
31  }
32
33  void ThermalManagerClass_Test::initialize(void)
34  {
35      manager->initialize();
36      CPPUNIT_ASSERT(manager->getCurrentThermalModel().getNodes().size() == TOTAL_NODES);
37  }
38
```

C++ source file length : 1347 lines : 53 Ln : 18

18 Uso de Macros Cppunit

Por otro lado y según se muestra en la imagen anterior (resaltado), en la definición de la clase se agrega el fixture especificado a un registro global de suites con la macro CPPUNIT_TEST_SUITE_REGISTRATION, en pos de disponer de todas las suites accesibles para el momento de la ejecución. La definición de la clase también implementa los métodos definidos en el header como por ejemplo el ThermalManagerClass_Test::initialize() (que verifica que se hayan cargado el total de nodos esperados al inicializar el modelo) haciendo uso de asserts como se comentó anteriormente.

La ejecución de los test se realiza con la herramienta TestRunner. Esta clase reproduce los test y obtiene información de sus resultados, indicando si un test fue exitoso, si falló o si se produjo una excepción.

19 Test Runner Cppunit

El anterior es el main desde donde se inicia la ejecución de los test, aquí se crea una instancia del TestRunner, se le asignan todos los Fixtures registrados globalmente y se llama al método run() para que los ejecute.

Para conocer otras macros de interés se puede consultar la página oficial del framework [RD59].

ii. Autotools

Autotools es un conjunto de herramientas que permiten crear paquetes de código portable de manera automática en sistemas Unix, reduciendo la complejidad del procedimiento y la cantidad de errores que pueden ocurrir al hacerlo manualmente [RD14].

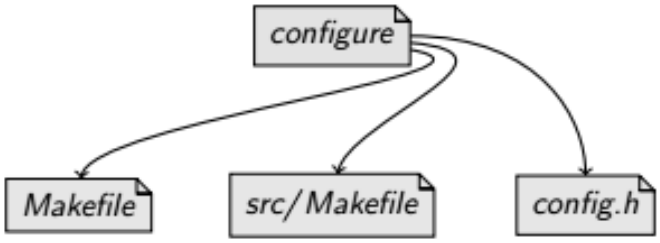
Antiguamente la dificultad de realizar paquetes portables, fue atacada con scripts. El objetivo de estas herramientas es que estos scripts sean generados automáticamente. Los paquetes resultado de este sistema de construcción (build) siguen el estándar GNU que abarca una serie de prácticas que aseguran la portabilidad indicando el modo en que se debería realizar el reporte de errores, obligando a contener el conjunto de opciones estándar en los comandos de línea, implica también un estilo de codificación, cierta configuración y convenciones para aplicar en los Makefiles.
Los proyectos que conforman Autotools son Autoconf, Automake y Libtool. Los cuales deben estar instalados en la máquina de desarrollo para generar paquetes distribuibles, una vez generado el paquete, el usuario final no requiere de las mismas para instalarlo.

Entre los aspectos positivos de estas herramientas se encuentran el soporte para compilación cruzada, su fácil lectura, poseen manejo de dependencias de headers, permiten crear un sistema de construcción con pocas instrucciones y centralizan las mejoras relacionadas a la portabilidad. Sin embargo aunque Autotools intenta simplificar el uso del sistema de construcción que propone el estándar GNU, también requiere de mantenimiento de la configuración para seguir los cambios que ocurran en el estándar.

Autoconf

Este proyecto surge del uso de un script para configuración, generalmente llamado “configure”, cuyo fin, es generar un sistema de construcción superando las diferencias que existen en las instrucciones de las plataformas Unix. En 1991 se inició el proyecto Autoconf, como solución a la necesidad mantener estos scripts para facilitar la portabilidad del código.

Autoconf es un paquete de macros M4 extensible que produce scripts Shell diseñados para determinar características específicas de la plataforma a partir de la descripción de algunas variables que indican dónde encontrar las librerías, dónde encontrar los headers, dónde se instalará el resultado de la compilación, etc. De manera que estos meta-scripts utilizan un lenguaje conciso basado en macros que permiten que la herramienta genere un script de configuración perfecto.



20 Archivos productos de la ejecución del script configure – Imagen recuperada de RD50

El resultado de este comando autoconf es el script "configure" cuya ejecución da como resultado el "config.h" y los makefiles para generar el paquete. El script config.h generado, es portable, correcto y más fácil de mantener y de depurar que una versión del mismo script realizada a mano [RD50].

Para generar scripts de configuración con Autoconf es necesario tener instalado GNU M4 (procesador de macros Unix). Estos scripts son autocontenidos, por lo cual los usuarios de los mismos no requieren de la instalación ni de Autoconf ni de GNU M4 [RD60].

configure.ac

```
1  AC_INIT([libthermalcore],[1.0])
2  AM_PROG_AR
3  AM_INIT_AUTOMAKE([-Wall -Werror foreign subdir-objects])
4  AC_CONFIG_MACRO_DIR([m4])
5
6  AM_DISABLE_STATIC
7
8  LT_INIT
9  AC_PROG_CXX
10 AM_PATH_PYTHON
11
12 AM_CXXFLAGS="-std=c++0x -Wall -Werror"
13 AC_SUBST([AM_CXXFLAGS])
14
15 AC_CONFIG_FILES([
16     libthermalcore.pc
17     Makefile
18     cpp/Makefile
19     test/Makefile
20     python/Makefile
21     python/__init__.py
22 ])
23
24 AC_OUTPUT
25
```

C++ source file length : 346 lines : 15 Ln : 8 Col

21 Inicialización de Autotools

El archivo “configure.ac” de la imagen anterior corresponde a la libthermalcore en su versión 1.0 como se describe en la macro AC_INIT que inicializa la herramienta autoconf en la primera línea. Con AM_INIT_AUTOMAKE se declara una lista de opciones para automake. LT_INIT agrega tests en el script “configure” para que la herramienta libtool reconozca características del host. AC_PROC_CXX determina que se utilizará el compilador C++. AM_CXXFLAGS permite configurar flags para el compilador. AC_OUTPUT cierra el script configure.ac.

Autoconf y Automake se complementan con aclocal que es una utilidad que surgió para dar apoyo a la falta de flexibilidad que pudieran tener estas herramientas. El método con el que se agregan macros al proyecto de Autoconf es a través del archivo aclocal.m4 en el mismo directorio que configure.ac. Autoconf lo lee automáticamente mientras procesa el configure.ac [RD49].

Automake

El objetivo de Automake es convertir el proceso de construcción de un proyecto en una sintaxis de makefile estándar que trabaje correctamente y proporcione todas las funciones estándar que se esperan de un proyecto de software libre. Esta herramienta al igual que la anterior, simplifica el trabajo de tener que realizarlo manualmente.

Automake genera plantillas de makefile estándar (llamadas Makefile.in) a partir de archivos de especificación de compilación de alto nivel (denominados Makefile.am). Incluye manejo de dependencias.

Makefile.am

```
1  lib_LTLIBRARIES = libthermalcore.la
2  libthermalcore_la_includedir = $(includedir)/thermalcore
3  libthermalcore_la_include_HEADERS = \
4      include/ThermalManagerClass_c.h \
5      include/ThermalModelClass_c.h \
6      include/ThermalNodeClass_c.h \
7      include/NodesRelationClass_c.h \
8      include/AbstractModelInitializerClass_c.h \
9      include/ThermalHarnessClass_c.h \
10     include/ShadeParameterClass_c.h \
11     include/SurfaceClass_c.h \
12     include/IDissipative_c.h \
13     include/ISensor_c.h \
14     include/Mocks.h
15 libthermalcore_la_SOURCES = \
16     src/DBModelInitializerClass_c.h \
17     src/ObjectModelInitializerClass_c.h \
18     src/ThermalManagerClass_c.cpp \
19     src/ThermalModelClass_c.cpp \
20     src/ThermalNodeClass_c.cpp \
21     src/ThermalHarnessClass_c.cpp \
22     src/NodesRelationClass_c.cpp \
23     src/DBModelInitializerClass_c.cpp \
24     src/SurfaceClass_c.cpp \
25     src/ShadeParameterClass_c.cpp \
26     src/ObjectModelInitializerClass_c.cpp \
27     src/ThermalSolverClass_c.h \
28     src/ThermalSolverClass_c.cpp
29 libthermalcore_la_LIBADD = \
30     $(top_builddir)/cpp/src/ConfManager/libconfmanager.la \
31     $(top_builddir)/cpp/src/EventLogger/libeventlogger.la \
32     $(BOOST_THREAD_LIB) \
33     $(BOOST_SYSTEM_LIB) \
```

C++ source file length : 346 lines : 15 Ln : 8 Col

22 Makefile.am Autotools

El archivo “Makefile.am” que se muestra en la imagen anterior, corresponde a la Libthermalcore como se indica en la macro lib_LTLIBRARIES que es una lista de las librerías libtool que se desean construir.

En las macros la_include_HEADERS y _la_SOURCES se indican los .h y .cpp del proyecto respectivamente.

En _la_LIBADD se especifican las dependencias con otras librerías .la. En todos los casos se puede observar el uso del prefijo con el nombre de la librería a compilar.

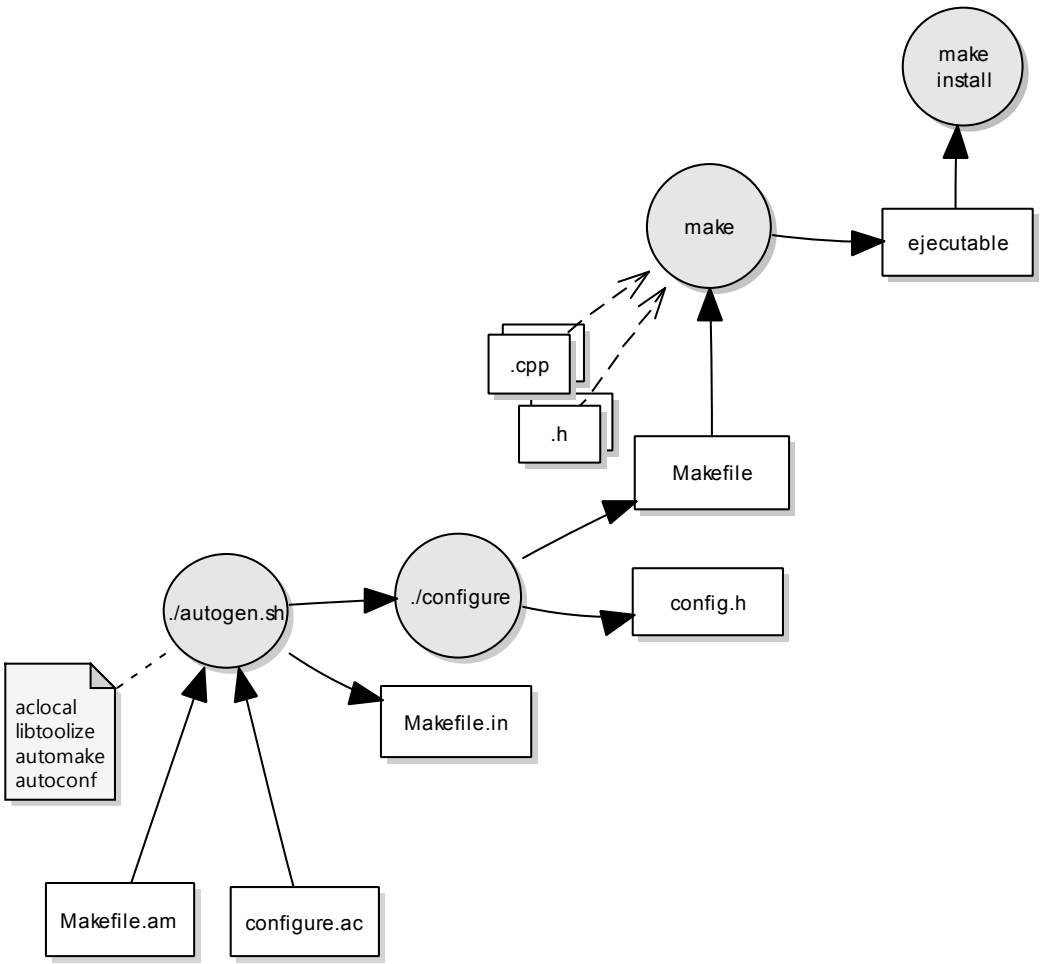
Libtool

Es una herramienta necesaria para proyectos que generan librerías. Ayuda a simplificar la construcción e instalación de librerías compartidas en sistemas Unix dado que muchos de ellos manejan las librerías compartidas de manera diferente: algunos las nombran libXX.so, otros XX.o. Es por ello que existe el concepto de librerías libtool con extensión .la que puede hacer referencia a una librería dinámica o estática [RD31].

Para inicializar la herramienta es necesario modificar el configure.ac para que incluya LT_INIT y el Makefile.am para que contenga la declaración de lib_LTLIBRARIES con la extensión .la. y la declaración de _LIBADD en el caso de contener dependencias con otras librerías [RD48].

Cadena Autotools

La cadena Autotools es la forma en que se denomina al orden que debe respetarse para la ejecución de los proyectos Autoconf, Automake y Libtool.
En la imagen a continuación se puede ver la secuencia de pasos a seguir para crear un proyecto Autotools:



23 Cadena Autotools

- Paso 1: ./autogen.sh

Este script contiene comandos del proyecto Autoconf que permiten la creación del script ./configure y Makefile.in. Los Makefile.in son templates que servirán de base para la creación de los Makefile.

Cada vez que se editan los archivos de entrada en la cadena se deben regenerar los archivos de salida, es por ello que este archivo reúne los comandos necesarios y debe ejecutarse cada vez que sea necesario regenerar los archivos Makefile.in y ./configure.

Los comandos que contiene el ./autogen.sh de estos proyectos son:

  - aclocal: escanea el archivo configure.ac para identificar el uso de macros de terceros y recopilar definiciones en aclocal.m4
  - libtoolize: provee una manera estándar de agregar soporte de Libtool al paquete
  - automake: crea los Makefile.in desde los Makefile.am y configure.ac
  - autoconf: crea el ./configure a partir del configure.ac

- Paso 2: ./configure

Este script generado automáticamente, va a verificar en el entorno de construcción la existencia de dependencias necesarias, generando Makefiles para la compilación y el archivo config.h mencionado anteriormente.

Su contenido está formado por variables de configuración estándar:

  - AM_CXXFLAGS, CXX (comandos de compilación C++)
  - CXXFLAGS (flags del compilador C++)
  - LDFLAGS (flags del linker)
  - CXXFLAGS (flags del preprocesador C++)

- Paso 3: make

La utilidad make automáticamente determina qué piezas de un programa necesitan ser recompiladas y emite comandos para recompilarlas, como resultado de ello se obtiene el componente de SW a parte de los Makefiles generados en el paso anterior de la cadena.

- Paso 4: make install

El make install es equivalente a realizar los comandos 'make install-exec' que se encarga de instalar los archivos dependientes de la plataforma y 'make install-data' que instala los archivos que no dependen de la plataforma. Luego de ejecutar el make install, se puede correr el comando 'ldconfig' que fue desarrollado en la sección "Desarrollo modular a través de librerías" y sirve para cargar librerías compartidas.

Como indica la cadena, ante un cambio en los archivos de entrada de alguno de los scripts, es necesario volver a ejecutar ese script y los sucesivos en la cadena.
Además conociendo el propósito de cada paso, es posible comprender los reportes de error e identificar en cuál de los archivos de entrada se encuentra. Por ejemplo si se agrega un nuevo archivo en el Makefile.am se debe regenerar el Makefile.in y el Makefile para poder compilar con el nuevo cambio.

iii. GUI en Python

Para diseñar la GUI del STS hecho en C++ se optó por utilizar Python. Este lenguaje posee una sintaxis OO, módulos, clases, excepciones, tipos de datos de alto nivel y tipos dinámicos. Es útil como lenguaje de extensión para aplicaciones codificadas en otros lenguajes que necesitan scripts sencillos de usar o automatización de interfaces. Al ser un lenguaje interpretado se evita el ciclo de codificación: editar/compilar/linkear/ejecutar. Además existen herramientas que permiten crear módulos de extensión que encapsulen librerías C++ existentes. Es por ello que se decidió que Python era el lenguaje que podría ser la unión para crear una nueva aplicación con interfaz gráfica STSGUI, a partir de las librerías libthermalclientapi y libstsclientapi.

Otro punto influyente en esta decisión fue el know how sobre interfaces gráficas provisto por los desarrolladores de simuladores anteriores que aportaron un proyecto base con componentes gráficos que interactuaban con la interface ISimulator que se mencionó en secciones anteriores.

Para lograr dicha GUI se empleó el framework PyQt4 que provee enlaces Python para GUI's en Qt. Es multiplataforma y posee múltiples licencias (GPL2, GPL3 y comercial). Cuenta con clases para implementar interfaces gráficas, manejar XML, implementar comunicación en red, manejar bases de datos SQL, y otras tecnologías que Qt también proporciona [RD34].

Además posee una jerarquía de clases 'widgets' que usan el mecanismo de señales y slots para comunicación entre objetos al igual que en Qt. Una señal se emite ante un evento en particular y un slot es la función definida para responder a tal evento. Esta técnica facilitó la creación de componentes de software reutilizables.

PyQt4 combina el poder de Qt y permite explotarlo con la simplicidad de Python [RD35].

Algunas de las desventajas que plantea la comunidad Python Argentina sobre esta herramienta según [RD36] son:

- No viene preinstalado con Python, se debe instalar por separado
- En ocasiones emerge la implementación en C++ subyacente, teniendo que hacer casteos entre tipos de datos, etc. El prefijo Qt/Q (QtGUI, QWidget, QApplication) hace el código más dependiente de tipos.
- No hay mucha documentación específica del framework

Entre los principales componentes que posee PyQt4 se utilizaron:

- QtCore

  Es un módulo que contiene las clases que no son GUI incluyendo el loop de eventos y el mecanismo de señales y slots, también incluye abstracciones independientes de la plataforma para hilos, archivos, memoria compartida, entre otros.

- QtGui

  Este módulo contiene las clases para interfaces gráficas.

También PyQt4 provee la posibilidad de usar el Qt Designer que es una herramienta gráfica para diseñar interfaces y generar código Python. Esto da la capacidad de separar el diseño de la interface, utilizando un compilador pyuic para componer las clases Python como se explica a continuación.

QtDesigner

Permite diseñar widgets, diálogos o ventanas principales completas usando formularios que aparecen en la pantalla o drag-and-drop. Permite pre visualizar los diseños y de esta manera generar prototipos con el usuario.

Esta herramienta almacena los diseños en archivos XML .ui y no genera código por sí misma, sino que incluye una utilidad que genera el código de la interfaz y la carga. En el caso de utilizarlo con PyQt4 esa utilidad es el módulo python 'uic' que carga los archivos .ui y genera código Python que crea las interfaces. Este módulo posee la interfaz por línea de comando pyuic.

El código generado se estructura como una clase simple que deriva del tipo Object en Python, cuyo nombre empieza con el prefijo Ui_ y posee el método setupUi() [RD37] que espera como parámetro el widget en el que se dibujará la ventana principal:

MainWindow.py

```
12  import sys
13  from PyQt4 import QtCore, QtGui
14  from PyQt4.QtCore import *
15  from PyQt4.QtGui import *
16  from ui.ui_dialog_MainSTS import Ui_dialog_MainSTS as gui
17  from stsclientapi.STSCLIENTAPI import *
18  from thermalcoreclientapi.THERMALCORECLIENTAPI import *
19  from Constants import *
20  from Configuration import Configuration
21  import datetime
22  import time
23  import PreferencesDialog
24  import ThermalDialog
25
26  class MainWindow(QMainWindow, gui):
27      def __init__(self, parent=None):
28          QMainWindow.__init__(self, parent)
29          self.setupUi(self)
30          desktop = QApplication.desktop();
31          self.screen_width = desktop.width();
32          self.screen_height = desktop.height();
33          self.sim = SimulatorXmlRpcClient_c()
34          conf = Configuration()
35          conf.LoadConfiguration()
36          self.sim.SetURL(Configuration.DSS_XMLRPC_SERVER_URL)
37          self.manager = ThermalManagerXmlRpcClient_c(Configuration.DSS_XMLRPC_SERVER_URL);
38          self.model = ThermalModelXmlRpcClient_c(Configuration.DSS_XMLRPC_SERVER_URL);
39          self.connect(self.pushButtonPlay, SIGNAL("pressed()"), self.OnPushButtonPlayPressed)
40          self.connect(self.pushButtonPause, SIGNAL("pressed()"), self.OnPushButtonPausePressed)
```

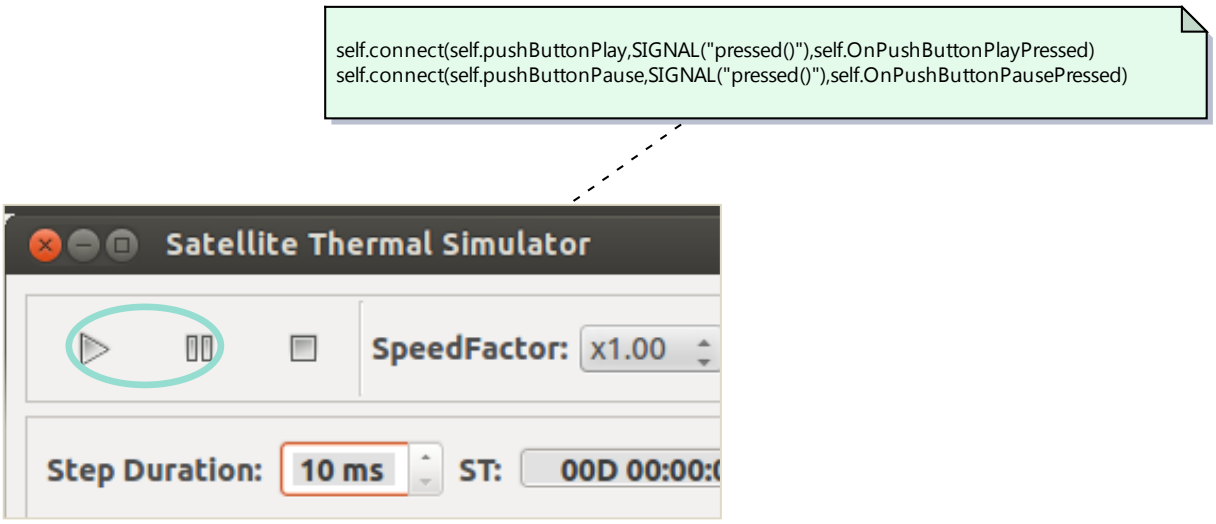
Python file | length : 10135 lines : 258 | Ln : 26 Col : 1 Sel : 0

24 MainWindow STSGUI

La imagen anterior muestra parte del script que se llama al ejecutar la STSGUI (ver especificaciones de arranque en la sección “Instalación del STS”).

En esta parte del script se puede observar que se importan los componentes QtCore y QtGui, el código Ui_díalog_MainSTS generado con QtDesigner y pyuic, las interfaces libthermalcoreclienapi y libstsclientapi wrappeadas con SWIG, entre otras cosas.

En la línea 28 se instancia la clase MainWindow que será la encargada de instanciar un objeto Ui_díalog_MainSTS, ejecutar el método setupUi(), crear los objetos relacionados con la simulación en las líneas 33, 37 y 38 y conectar las señales con las funciones de la interfaz del simulador que serán llamadas al producirse el evento, por ejemplo:

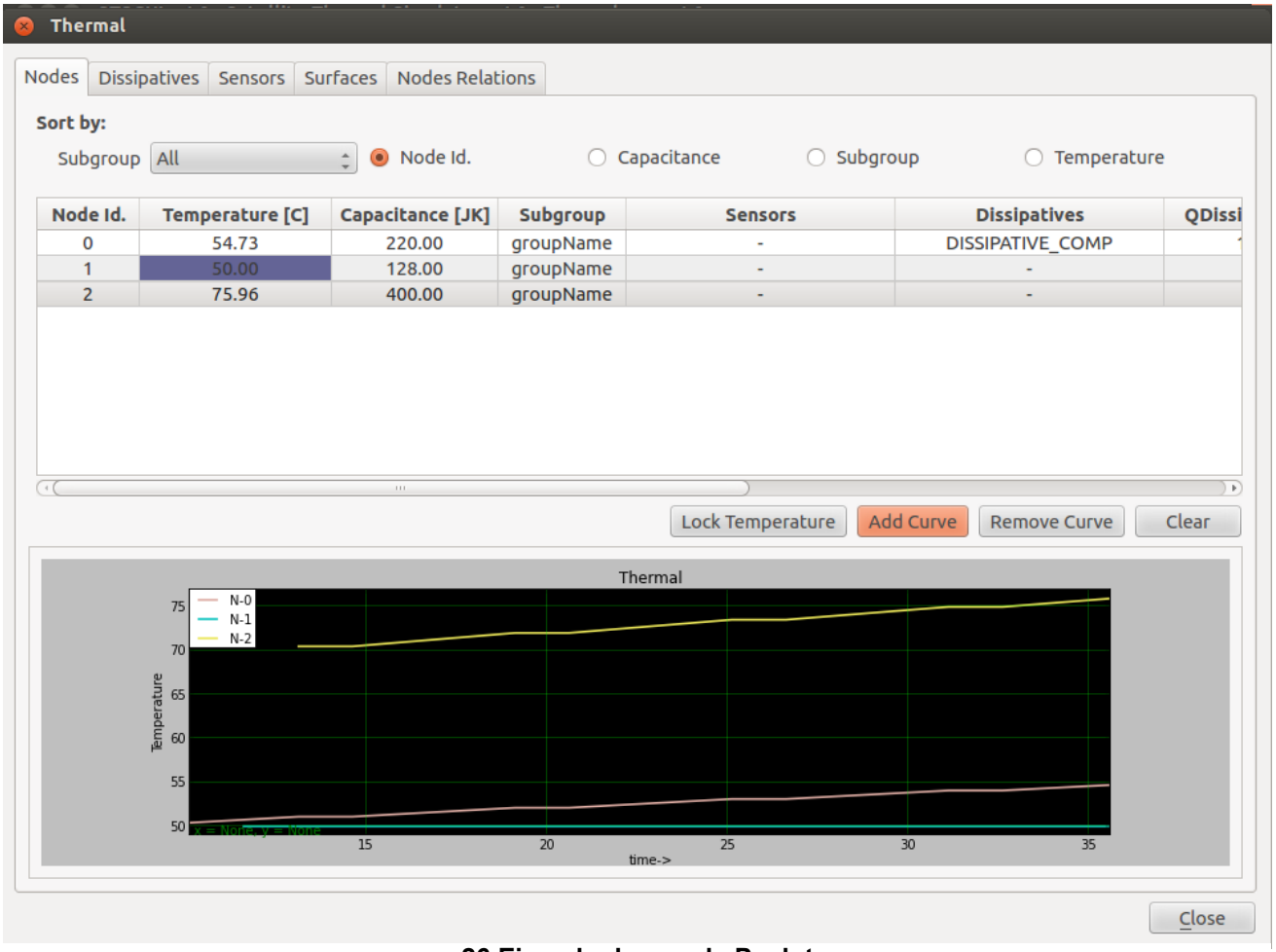


25 PyQt señales y slots

PyPlot

La librería PyPlot forma parte de la matplotlib que es una librería para hacer plots 2D de arreglos en Python, y surge con la idea de emular MATLAB desde este lenguaje, es por ello que usa NumPy que es un paquete de cálculo científico de Python [RD39].

Cada función de PyPlot realiza algún cambio en el gráfico (crea líneas en el plot, decora el plot con labels). En el caso del STS se utiliza un gráfico de series para hacer un seguimiento visual de la curva de temperatura de los nodos seleccionados y facilitar su comparación. El plot que se muestra a continuación, se actualiza siguiendo la velocidad de la simulación.



26 Ejemplo de uso de Pyplot

El eje ‘x’ representa el tiempo y el eje ‘y’ los valores de temperatura de los nodos elegidos para graficar. En este caso se puede observar que el nodo 0 se corresponde a la línea rosa, el nodo 1 que se encuentra lockeado en un valor fijo corresponde a la línea verde y el nodo2 con mayor temperatura que los otros se asocia a la línea amarilla unos centígrados más arriba. El gráfico en este caso permite detectar rápidamente que un nodo tiene temperatura constante (porque esta lockeado, porque no tiene relaciones con otros nodos, o que el componente que representa está roto) y que hay otro que se está sobrecalentando (por incidencia directa del sol o por algún mal funcionamiento electrónico simulado).

iv. Desarrollo de API XML-RPC

XML-RPC es un protocolo estándar y portable de comunicación entre computadoras a través de llamadas remotas de procedimiento sobre HTTP. Funciona bajo el modelo cliente-servidor. En las llamadas remotas el cliente hace peticiones aisladas al servidor, que recibe el pedido y responde. Este protocolo fue diseñado por Dave Winer de la empresa UserLand Software en asociación con Microsoft, para lograr que sea simple la transmisión y procesamiento de los tipos de datos. Soporta: int, string, boolean, double, datetime, array, struct, binarios con base64 bits.

Para ello codifica la información en pedidos y respuestas XML, fácilmente legibles. La desventaja de este formato es que usa más recursos de la red que otros tipos de protocolos [RD28].

Las librerías implementadas (STS y LibThermalCore) se comunican a través de este protocolo, permitiendo que el servidor y el cliente se encuentren corriendo en distintas máquinas de la red. Como se mencionó con anterioridad se desarrollaron las librerías servidor: libthermalserverapi y libstsapi; y las librerías cliente: libthermalclientapi y libstsclientapi respectivamente. La STSGUI como interfaz gráfica que consulta los datos del modelo tiene como dependencias ambas librerías cliente.

Parte de la motivación para seleccionar este protocolo fue su simpleza. Se utilizaron como dependencias las librerías Xmlrpc-c que proveen soluciones para el problema de la comunicación entre librerías en C++. A continuación se muestra una lista de las mismas [RD28]:

- Lib Xmlrpc++

  Provee facilidades generales de XML-RPC como la codificación y decodificación de XML (no específicas de clientes o de servidores).

- Libxmlrpc_client++

  Proporciona funcionalidad para implementar el cliente

- Libxmlrpc_server_abbys++

  Proporciona funcionalidad para implementar un servidor basado en abyss HTTP server.

Existen otras librerías con igual funcionalidad que Xmlrpc-c. Según [RD30] se muestra una pequeña comparación:

- XML-RPC vs. CORBA (Common Object Request Broker Architecture)

  De manera similar al protocolo elegido, CORBA se utiliza para aplicaciones distribuidas, multicapas y OO.

  Fue creada por Gnome. Tiene soporte a varios lenguajes en particular a Java y C++.

  Sin embargo en comparación, es más complejo y su curva de aprendizaje más lenta.

- XML-RPC vs. DCOM (Distributed Component Object Model)

  Fue creado por Microsoft, para sistemas Microsoft que se comunican en red usando formatos binarios propietarios.

- XML-RPC vs. SOAP (Simple Object Access Protocol)

  Es muy similar a XML-RPC en cuanto a que permite realizar llamadas a procedimientos en formato XML con el protocolo HTTP. Sin embargo XML-RPC puede ser descripto como un subconjunto de la funcionalidad SOAP, ya que SOAP puede correr sobre protocolos que XML-RPC no implementa y posee otras implementaciones de seguridad (WS-Security

además de HTTP AUTH por ejemplo). La principal ventaja de SOAP sobre XML-RPC es que soporta especificaciones WSDL (Web Services Description Language) lo que facilita el descubrimiento y la integración con un servicio web remoto [RD38]. Actualmente lo mantiene el grupo de trabajo de W3C (World Wide Web Consorsium).

Un inconveniente que surgió durante el desarrollo de la librería cliente de la funcionalidad térmica fue relacionado a los tipos de datos. En una de las pantallas en que se veía evolucionar la temperatura, la trasmisión de la respuesta en xml desde el servidor fallaba aleatoriamente. Luego de depurar el problema se detectó que este error ocurría al intentar dar formato de 2 decimales (.2F) a un NaN (Not a Number). La causa de este NaN era que los datos del modelo térmico eran incorrectos, por lo que las temperaturas tendían al infinito con el paso del tiempo convirtiéndose en un tipo de dato no representable por la librería XML-RPC. Dado que esta situación es posible en un escenario de uso común del STS, se optó por verificar si el valor es NaN antes de formatearlo, en ese caso se eligió un valor muy alto preestablecido para representar que dicha temperatura se fue al infinito.

v. Extendiendo la API

Se decidió optar por una herramienta que extienda automáticamente la funcionalidad publicada por las librerías clientes C++ a través módulos Python que faciliten dicha interacción. La importancia de tener la posibilidad de tener acceso a la librería a través de un lenguaje de scripting se detalló en la sección anterior. Existe una lista variada de herramientas que ofrecen esta funcionalidad, por ejemplo: SWIG, GRAD, bgen, etc. Para este trabajo se eligió SWIG (Simplified Wrapper and Interface Generator) que crea módulos automáticamente.

Las relaciones se definen típicamente según archivos de interface .i. Este archivo contiene las declaraciones de las clases, enumerativos, tipos C++ que se desean acceder en el script Python y directivas SWIG precedidas de '%' [RD13].

THERMALCORECLIENTAPI.i

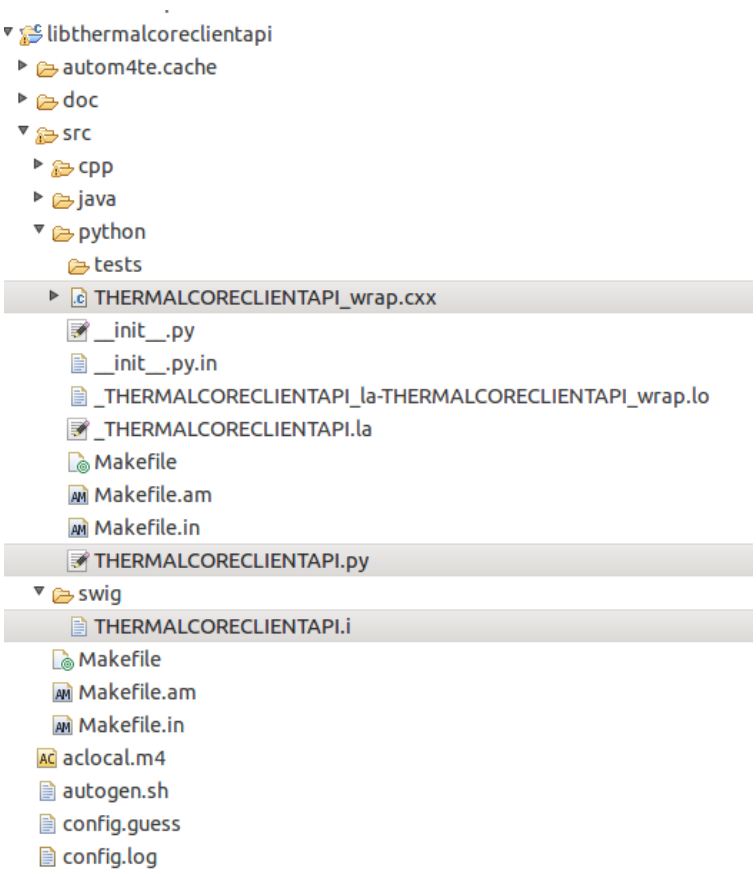
```
%module(package="thermalcoreclientapi") THERMALCORECLIENTAPI
%include "std_string.i"
%include "std_vector.i"
%include "std_map.i"
%include "exception.i"
%include "stdint.i"
%include "cdata.i"
%{
#include "cpp/AbstractXmlRpcClient_c.hpp"
#include "cpp/ClientStructTypes.hpp"
#include "cpp/ThermalNodeClassClient_c.hpp"
#include "cpp/ClientMockComponent.hpp"
#include "cpp/ClientEnum.hpp"
#include <dsscommon/Exception_c.hpp>
#include "cpp/ThermalManagerXmlRpcClient_c.hpp"
#include "cpp/SurfaceClassClient_c.hpp"
#include "cpp/ThermalModelXmlRpcClient_c.hpp"
%}
namespace std {

   %template(IntVector) vector<int>;
   %template(DoubleVector) vector<double>;
   %template(StringVector) vector<string>;
   %template(UIntVector) vector<unsigned int>;
   %template(ByteVector) vector<unsigned char>;
   %template(UShortVector) vector<unsigned short>;
   %template(vector_HarnessClient) vector<ThermalHarnessClient_t>;
   %template(vector_NodesRelationClient) vector<NodesRelationClient_t>;
   %template(vector_MockDissipativeClient) vector<MockDissipativeClient>;
   %template(vector_MockSensorClient) vector<MockSensorClient>;
```

thon file length : 1973

27 Ejemplo de uso de SWIG

La directiva %module especifica el nombre del módulo que se desea crear en Python. El código que se encuentre dentro de %{… %} se copia dentro del código wrapper generado por SWIG, este código por lo general hace referencia a archivos header [RD32].

Si se especifica `%module THERMALCORECLIENTAPI', todo será wrappeado en el módulo Python 'THERMALCORECLIENTAPI' y se genera el código del wrapper automáticamente en el archivo 'THERMALCORECLIENTAPI.py' y una extensión a bajo nivel 'THERMALCORECLIENTAPI.so'.



28 Directorio del proyecto

La extensión Python se compila dentro de la librería C++. Se utilizan los comandos indicados en la línea coloreada del makefile (ver imagen a continuación) de la libthermalclientapi que se encuentra en la carpeta python como se muestra en la imagen anterior del directorio del proyecto:

```
swig -c++ -python THERMALCORECLIENTAPI.i
```

Makefile.am

```
1  SWIG = swig3.0
2  SWIG_INTERFACE = $(top_srcdir)/src/swig/THERMALCORECLIENTAPI.i
3
4  pythermalcoreclientapidir = $(pyexecdir)/thermalcoreclientapi
5
6  pythermalcoreclientapi_LTLIBRARIES = _THERMALCORECLIENTAPI.la
7  pythermalcoreclientapi_DATA = __init__.py THERMALCORECLIENTAPI.py
8
9  _THERMALCORECLIENTAPI_la_SOURCES = THERMALCORECLIENTAPI_wrap.cxx
10 _THERMALCORECLIENTAPI_la_CPPFLAGS = `python-config --includes` -I$(top_srcdir)/src
11 _THERMALCORECLIENTAPI_la_LIBADD = $(top_builddir)/src/cpp/libthermalcoreclientapi.la
12 _THERMALCORECLIENTAPI_la_CXXFLAGS = -fno-strict-aliasing -std=c++0x
13
14 THERMALCORECLIENTAPI_wrap.cxx: $(SWIG_INTERFACE)
15     $(SWIG) -c++ -python $(_THERMALCORECLIENTAPI_la_CPPFLAGS) -outcurrentdir $(SWIG_INTERFACE)
16
17 BUILT_SOURCES = THERMALCORECLIENTAPI_wrap.cxx
18 CLEANFILES = THERMALCORECLIENTAPI_wrap.cxx __init__.py THERMALCORECLIENTAPI.py
19
```

29 Ejemplo de uso de SWIG

Luego de la compilación se obtiene el archivo fuente C++ THERMALCORECLIENTAPI_wrap.cxx y el ya mencionado THERMALCORECLIENTAPI.py como puede observarse en la imagen que brinda un panorama del directorio de la librería cliente. El archivo C++ generado es el que debe ser compilado y linkeado por el resto de la aplicación para crear el módulo de extensión .py que es el que se importa desde los scripts.

El nombre de estos archivos deriva del nombre del archivo .i que se utiliza para especificar la interface. Por lo tanto para el THERMALCORECLIENTAPI.i se creará THERMALCORECLIENTAPI_wrap.cxx. El nombre del archivo .py será el que se especifica en la directiva %module [RD33].

Para usar el módulo creado se utiliza la sentencia 'import' y Python carga e inicializa el módulo en la marcha. El siguiente es un ejemplo de un script que imprime los tiempos de una determinada simulación:

TestSimulatorMethods.py

```
1  from stsclientapi.STSCLIENTAPI import *
2  from time import sleep
3
4  print "Configuring..."
5  sim = SimulatorXmlRpcClient_c();
6  sim.SetURL("http://localhost:8890/RPC2");
7
8
9  for i in range(1):
10     print "Play Simulation..."
11     sim.Run()
12     sleep(1)
13
14     print "simulation isAlive: " + str(sim.IsAlive())
15     print "simulation time: " + str(sim.GetSimulationTime())
16     print "simulation time string: " + sim.GetSimulationTimeString()
17     print "simulation Mission time: " + str(sim.GetMissionTime())
18     print "simulation Mission time string: " + sim.GetMissionTimeString()
19     print "simulation epoch time: " + str(sim.GetEpochTime())
20     print "simulation epoch time string: " + sim.GetEpochTimeString()
21     print "simulation state: " + sim.GetSimulationState()
22
23     dateET = sim.GetEpochTimeYMDHMS()
24     print dateET
25     #print dir(dateET)
26
27     print "simulation Zulu time: " + str(sim.GetZuluTime())
28     print "simulation Zulu time string: " + sim.GetZuluTimeString()
29
30     dateZT = sim.GetZuluTimeYMDHMS()
```

Python file length : 2812 lin

30 Testing de la API

En la primera línea se cargan todos los nombres definidos en la interface wrappeada en Python (stsclientapi) que se quieren utilizar.

Errores comunes en la codificación y uso de estos módulos de extensión son:

- no coincidir el nombre del módulo del import con el de la directiva %module
- no haberlo compilado aun por lo que no se dispone de los wrappers
- que falte wrappear los tipos customizados que utiliza el módulo

vi. Control de Versiones

Las herramientas de control de versiones se encargan de registrar cambios en los archivos que conforman los proyectos de SW de manera que nos permiten remitirnos a alguna versión anterior de los mismos cuando sea necesario. En los proyectos de simulador y librería térmica se utilizó GIT para tal fin.

La selección de esta herramienta frente a Subversion SVN, surge del análisis de comparativas recopiladas de la web, teniendo en cuenta el flujo de trabajo distribuido que el equipo de desarrollo de simuladores posee. En dicho grupo se participó durante la implementación del STS y la Libthermalcore.

La dispersión en la que se almacenan los datos, es la mayor diferencia que se encuentra entre las posibles opciones de herramientas de control de versión. Esto significa que existen las herramientas CVCS (Centralized Version Control Software) cuyo repositorio es centralizado y las DVCS (Distributed Version Control Software) con repositorios distribuidos.

En cuanto a las opciones de repositorio centralizado como es el caso de Subversion SVN, tienen como principal desventaja que el servidor puede representar un único punto de falla.

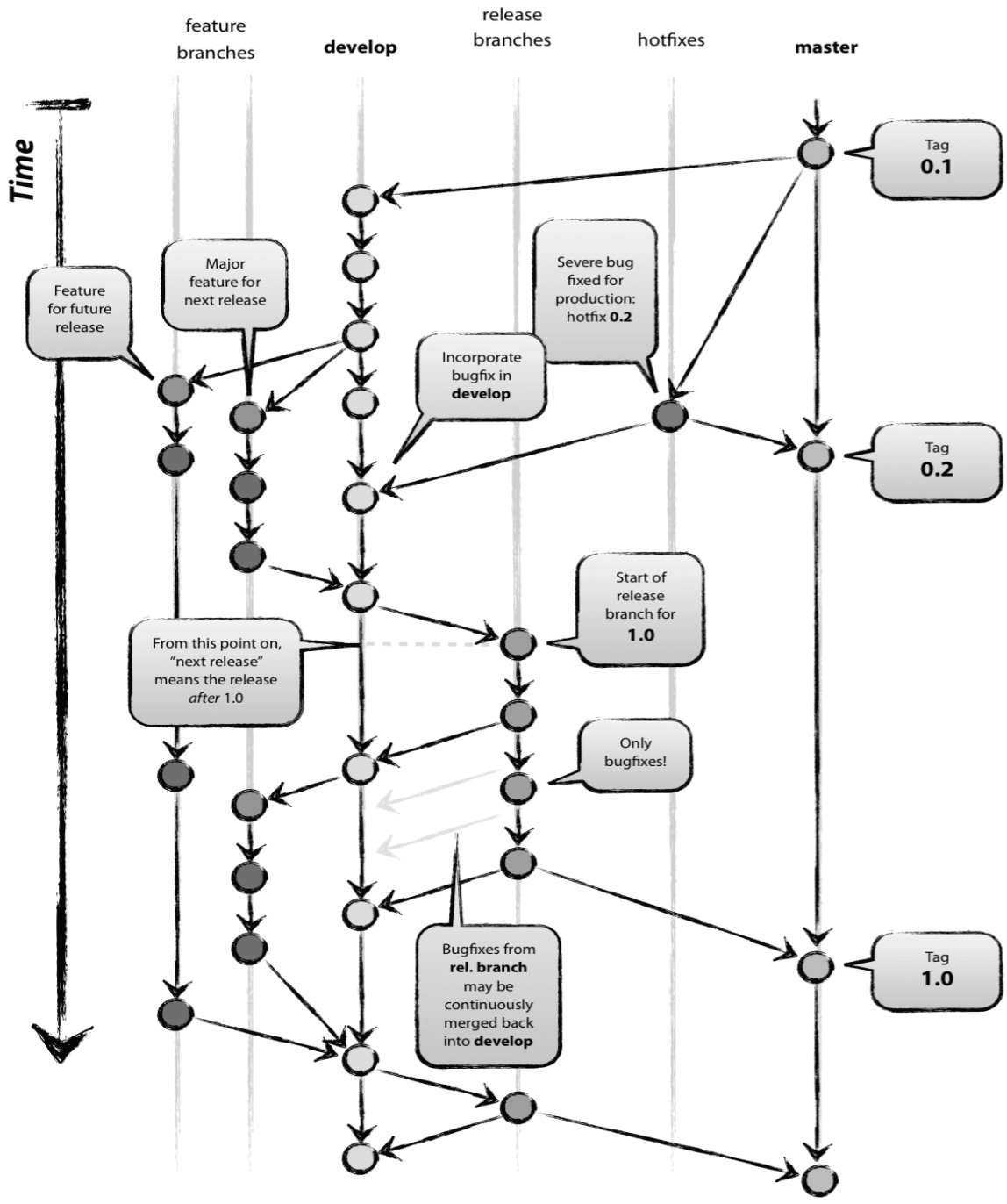
En el control de versión distribuido como es el caso de GIT (y otros como Darcs, Mercurial y Bazaar) cada cliente posee una copia del repositorio y no sólo su ultimo estado. Este repositorio local, permite al desarrollador trabajar con diferentes branches localmente o mantenerse al día respecto del repositorio compartido. Esta característica puede transformarse en una desventaja ya que cada cliente debe tener capacidad suficiente de almacenamiento para contener el repositorio completo que en algunos casos implica unos cientos de gigabytes. Una medida para evitar esta situación es dividir los proyectos en pequeños repositorios GIT para que permita la descarga del subconjunto requerido. Aunque también esta solución incluye un mantenimiento adicional, la perdida de visibilidad del código completo y de la posibilidad de realizar Refactorings a gran escala de manera atómica.

GIT no posee control de acceso exclusivo por archivo, como existe en SVN. En SVN esta posibilidad de dar autorización para leer o modificar archivos en los repositorios se configura de manera sencilla mediante la variable authz-db en svnserve.conf con la dirección del archivo que define las reglas de acceso y flexibiliza la implementación de roles de los usuarios.

Entre los inconvenientes de SVN podemos mencionar que solo permite el merge entre dos branches a la vez, al hacer merge del contenido del branch con el trunk, obligatoriamente el código queda compartido en un repositorio general, esta situación a diferencia de GIT, impide realizar commits de experimentación con nuevas funcionalidades e imposibilita almacenar en el repositorio funcionalidad incompleta. SVN permite realizar merge entre arboles no relacionados totalmente, esta característica hace del merge una operación propensa a errores [RD62].

En contraposición, GIT permite realizar merge de más de 2 branches a la vez, permite borrar branchs obsoletos manteniendo su historial, posee mejor seguimiento de archivos renombrados en comparación con el SVN y promueve con su forma de uso algunas buenas prácticas como branch por funcionalidad, branches locales para experimentación de manera que no se utilice espacio en el repositorio público, etc.

Otra de las diferencias surge del modo en que ambas herramientas representan los datos. En el caso de Subversión, las revisiones son deltas de un estado inicial y el servidor debe resolver esos deltas para obtener el estado concreto de la revisión. De este modo los branchs son copias livianas del árbol de trabajo. GIT por su parte no representa las diferentes revisiones con deltas sino como snapshots del proyecto total ya que los mantiene como repositorio local, dando lugar a una de sus mayores ventajas que es su modelo para ramificar el trabajo [RD63]. GIT permite tener múltiples ramas de trabajo (branchs) almacenadas localmente e independientes entre sí, y provee operaciones de unión y borrado para las mismas. Esto permite probar una idea en un nuevo branch sin modificar el estado del código en desarrollo, crear una nueva rama para implementar una funcionalidad y luego unirla al código principal una vez terminada. Además es posible seleccionar cuáles de esos branches serán compartidos al guardarlos en el repositorio remoto.



31 Modelo de uso de branches – Imagen recuperada de RD61

En [RD61] se detalla una buena convención y semántica para los branchs que fue utilizada al momento de trabajar con GIT en el proyecto STS. Este modelo propuesto en la imag en anterior, sugiere utilizar dos branchs principales master y develop, el primero con las releases listas para poner en producción y el segundo para integración de funcionalidades de la siguiente entrega. Además indica convenciones de nombres, secuencias de comandos y buenas prácticas para los casos en que se desee crear branchs para desarrollo de nuevas funcionalidades, para solucionar bugs de versiones en producción y, en el caso de los release branchs, para preparar una nueva versión de producción, incluyendo la solución de errores menores y la preparación de metadatos (número de versión, fechas de compilación, etc.); mientras que en develop se continua con funcionalidades para la siguiente release.

Se utilizaron en el caso de estudio features branchs, master branch y develop branch según la convención propuesta.

Capítulo 4: Administración del Proyecto

El propósito de esta sección es describir los temas relacionados con el Management del Proyecto STS, este análisis se realizó en una instancia inicial en la que se evaluaron diferentes características del producto a realizar y las diferentes etapas de su entrega.

i. Planificación temporal del proyecto

La planificación de un proyecto de SW implica la identificación de tareas, la asignación de tiempos, de recursos, y la programación de la secuencia de ejecución de las mismas, de forma que se detecten las que son críticas rápidamente y que a su vez permita que el tiempo de desarrollo del proyecto sea mínimo.

Como resultado de la fase de análisis, se identificaron los componentes involucrados en el desarrollo. Identificar los componentes sirvió de panorama preliminar para identificar los módulos a realizar. Estos módulos se listan a continuación:

- Librería que contenga un núcleo de cálculo térmico
- Librería que servirá de API XML-RPC para interactuar con los sistemas externos. Esta API debe tener bindings para python.
- Simulador que haga uso de la libSmp2, que implementa el estándar para tal fin. Esta reutilización de las clases que implementan el mapeo de interfaces del estándar, en lenguaje C++, reducirá el alcance y esfuerzo del desarrollo.
- Librería API XML-RPC del simulador, que publique funciones relacionadas con el control del tiempo en la simulación.
- Librería cliente XML-RPC del simulador, que consuma las funciones que éste pública.
- Librería cliente XML-RPC de la librería térmica, que consuma las funciones que ésta publica.
- Base de datos térmica.
- Base de datos de componentes del simulador.

Se decidió que el desarrollo se realizaría en tres etapas. Durante la primera se extraería como librería el modelo térmico, su api y la base de datos térmica, de manera que pueda ser testeada independientemente mediante scripts.

En la siguiente etapa del desarrollo se elaboraría el simulador previamente descripto con su interfaz de comunicación para su correspondiente testing e integración con la LibThermalCore. Con esta etapa se alcanzaría casi el 80% del desarrollo.

Y, finalmente se implementaría la GUI que muestre los datos de manera amigable.

En el primer análisis, no se identificó la necesidad de las dos librerías cliente (libstsclientapi y libthermalcoreclientapi) que permitieran consultar la API desde la interfaz gráfica (GUI), línea de comando (CLI), scripts, etc. Como tampoco se tuvo en cuenta que también era necesaria una BD dedicada al STS. En consecuencia estos módulos fueron subestimados.

Es importante mencionar que esta falencia en el análisis trajo aparejadas modificaciones en las estimaciones originales y por lo tanto, en el presupuesto total del sistema, ya que no se ponderaron correctamente. Se tratan estas cuestiones a lo largo de este capítulo.

ii. Memorias técnicas

Las memorias técnicas permiten dimensionar el impacto que tuvieron las decisiones de diseño en la planificación final del proyecto. Tomar la decisión de adoptar una nueva herramienta o decidir desarrollar una arquitectura más escalable puede implicar una demora de una semana de trabajo en pos de obtener mejores resultados. En algunos casos estas decisiones no se pueden prever y como consecuencia, se retrasan las entregas.

La siguiente es la lista de los riesgos identificados en un principio. Algunos de ellos, tuvieron incidencia en el avance del proyecto como se describe a continuación:

- Subestimación del tamaño del sistema

  En el planteo inicial de la solución a nivel de diagrama de clases, se realizó el diseño de las clases encargadas del cálculo térmico, abarcando la creación del modelo, su configuración y cálculo de temperatura de sus nodos. Sin embargo al momento de pensar el alcance de la librería térmica como parte de un simulador, tuvieron que tenerse en cuenta algunos detalles de interacción con el framework de la libsmp2, que se fueron conociendo a través de Ingeniería Inversa. Este proceso permitió comprender cómo fue la implementación del cálculo térmico dentro de un simulador tan amplio como es el SSS. También implicó tiempo extra en la resolución del problema ya que se optó por realizar Refactorings al código de cálculo térmico extraído del SSS, con el objetivo de separar responsabilidades, desacoplar las clases del dominio térmico de las que interactuaban con la interfaz del simulador y así obtener un código más legible. En este punto la solución vino de la mano del Handbook [RD25] que muestra las responsabilidades, interfaces y estados por los que pasa un simulador y cómo las clases del negocio deben implementar el estándar. Como resultado de este proceso de Ingeniería Inversa y refactorización se obtuvo un módulo térmico suficientemente abstracto y testeable.

- Subestimación de la cantidad de componentes a implementar

  Este caso se tuvo en cuenta en el Análisis de Riesgos, debido a que un componente nuevo implica más tiempo de capacitación, diseño, codificación, testing unitario y funcional para poder lograr un entregable estable. La estrategia de disminución propuesta era realizar un análisis que tuviese una visión general del dominio. Haberlo hecho cuando recién se comenzaba con el entendimiento del problema, no fue suficiente para mitigarlo; es decir que hubieron algunos componentes que quedaron sin un presupuesto correcto.

  En particular (como se mencionó en la sección Planificación temporal del proyecto) las librerías libthermalclientapi y libstsclientapi fueron subestimadas dado que implicó un esfuerzo extra de capacitarse respecto de la librería libxmlrpc_client++ y de diseñar las librerías cliente. Algo similar ocurrió con la base de datos dedicada del STS encargada de unir los datos del modelo térmico con componentes reales e identificables físicamente. Esta base de datos no se tuvo en cuenta en un comienzo y luego fue necesaria su creación y la implementación de la carga en memoria.

- Falta de compatibilidad entre el producto a generar con las librerías que tiene como dependencias

  Para tratar este riesgo la estrategia que se utilizó fue de anulación buscando utilizar la arquitectura necesaria para poder ser compatible con las dependencias, es por ello que las necesidades de este tipo se incorporaron como restricciones de diseño y se logró de esta manera evitar que exista alguna falta de compatibilidad con las dependencias.

- Falta de disponibilidad del entorno de desarrollo configurado o de herramientas de gestión del proyecto

El riesgo de no disponer del entorno de desarrollo configurado ocurrió. Generó una demora en el inicio de la codificación. El know how sobre configuración de ambiente existía, pero sobre maquinas Linux nativas, por lo tanto al querer llevarlo a una máquina virtual que corriera en Window7 implicó algunas demoras en la resolución de: acceso a la red con proxy, datos compartidos, permisos de usuarios, etc. La decisión de realizar el trabajo en una máquina virtual se basó en la necesidad disponer del proyecto y su entorno para consulta durante la posterior redacción de la tesina. Se prefirió utilizar la PC que ya disponía en el trabajo (en lugar de una notebook u otra PC con Linux nativo) ya que la misma tenía características de HW adecuadas para el simulador. La intención fue mantener Window7 para desarrollos futuros y evitar los pormenores de instalarle un arranque dual a la PC.

- Finalización fuera de plazos originales

Este riesgo fue uno de los que se concretó como consecuencia de los otros. Cada uno de estos imprevistos involucraron un esfuerzo no estimado. Dentro de la estrategia para disminuir el riesgo de finalización fuera de plazos originales, se había planteado la asignación de más recursos al proyecto, sin embargo por el nivel de avance del mismo era conveniente afrontar la demora en días, en lugar de inducir a un nuevo recurso en el dominio del sistema. La metodología para eliminar los obstáculos que surgieron en el avance del proyecto, fue realizar reuniones de revisión con más frecuencia.

- La detección de errores de arquitectura que impliquen volver a las fases de diseño

Este riesgo ocurrió a partir de errores de codificación. Como se mencionó anteriormente, se diagnosticaron memory leaks en una etapa avanzada de la implementación, se detectaron gracias a una herramienta de soporte (Valgrind) y se solucionaron. Esta tarea agregó aproximadamente una semana más a la entrega.

- Subestimación del tiempo requerido para cada una de las fases del proceso

Las siguientes son algunas mejoras que se fueron planteando conforme se iba avanzando en las iteraciones y profundizando sobre el diseño de cada nueva funcionalidad todas implicaron algún tiempo extra de diseño y evaluación de la solución.

Fue una importante decisión de diseño, agrupar los termistores, termocuplas, heaters, etc. según su similitud en el comportamiento del cálculo de su temperatura, quedando clasificados en ISensor e IDissipatives. Más detalles ver Apéndice B – SDD Capítulo 1.

También fue útil en el avance de la libthermalcore separar el acceso a los datos, de las clases del negocio; esto permitió realizar testing de la librería sin agregar la complejidad de una conexión a la base de datos a través de una clase generadora de un modelo base.

Luego, una vez codificada la clase de conexión a la BD (DBModelInitializerClass_c) se configuró esta nueva fuente de datos en los test del ThermalManagerClass_c de manera transparente.

Del mismo modo se separó el acceso a datos en el STS, donde se recreaban las cargas de resistencia continua, cargas de corriente y potencia continuas con objetos mock que implementaban la interface IDissipative y, por otro lado los termistores y termostatos fueron representados con mocks con la interfaz ISensor. Al finalizar la creación de la base de datos fueron reemplazados por los datos reales de forma transparente. Ver Apéndice B – SDD Capitulo 2.

Otra resolución en cuanto al diseño fue la creación de una jerarquía de servidores xml-rpc para que el simulador que incluya la librería térmica (en este caso el STS, pero podría ser otro)

pueda contar con la interfaz térmica y toda su funcionalidad simplemente heredando de la clase LibThermalXmlRpcServer_c. De esta manera se evitó la duplicación de código de registro de métodos xml-rpc de los servidores. Esta decisión puede verse en las librerías que implementan las APIs: libthermalserverapi y libstsapi.

- Falta de disponibilidad del cliente para reuniones con el equipo de desarrollo

Se decidió aceptar este riesgo en caso de que ocurriese. Se tenía en cuenta que era baja probabilidad y el impacto tolerable. Como resultado se contó con el cliente de manera satisfactoria.

- Subestimación de la curva de aprendizaje de las nuevas tecnologías a utilizar

La estrategia ideada fue de transferencia ya que proponía c*onsultar con especialistas en las nuevas tecnologías.* Este riesgo identificado pudo mitigarse parcialmente por lo que se vio afectado el calendario de entrega también por este motivo, a medida que iba surgiendo la necesidad, se asignó el tiempo necesario para el aprendizaje.

iii. Estimación y carga de trabajo real

La estimación es una actividad de la planificación que intenta determinar cuánto dinero, recursos y tiempo tomará construir un sistema. Una manera de ponderar estos ítems es sintetizarlos como el esfuerzo que demandaría la concreción del proyecto. El objetivo de la estimación es predecir las variables involucradas en el proyecto con cierto grado de certeza.

La estimación es un proceso continuo, dado que cuanto más se conoce el sistema, más variables hay disponibles para la estimación [RD52].

Algunas técnicas de estimación son:

- La opinión de los expertos, basada en la experiencia profesional
- La analogía con uno o más proyectos pasados, para ello deben ser similares los proyectos entre sí
- La descomposición del proyecto en componentes propone la estimación total como sumatoria de la estimación de las partes
- Las ecuaciones de estimación que a partir de fórmulas matemáticas establecen la relación de algunas medidas de entrada y determinan el esfuerzo que se requerirá. En [RD51] se detallan dos en particular, COCOMO y Puntos de Función comparando estimaciones para un proyecto en distintos lenguajes de programación.

Fue posible realizar una estimación general del esfuerzo que conllevaban el STS y la LibThermalCore, luego del modelado de requerimientos y análisis inicial en el que se dimensionó el alcance. La técnica utilizada fue una combinación de analogía con otros simuladores y la opinión de expertos. Es por ello que en base al conocimiento del alcance se realizó un presupuesto inicial por un desarrollo de 6 meses con un programador. A medida que se avanzó en las iteraciones incrementales se fue visualizando la necesidad de extender este plazo. Más adelante en este capítulo se detalla el tema.

La siguiente es la distribución del esfuerzo estimado a lo largo de la duración prevista inicialmente:

Actividades del Desarrollo de SW	Esfuerzo en horas	TOTAL %
Requerimientos	112	8%
Arquitectura	200	14%
Implementación y Pruebas	906	64%
Capacitación	112	8%
Planificación	70	5%
TOTAL	**1400**	**100%**

32 Primera estimación de esfuerzo del STS

Para lograr estos porcentajes se tomaron como referencia la distribución del esfuerzo de experiencias anteriores. Es decir que en base a la duración prevista se asignó cierto porcentaje a cada fase para obtener la cantidad de horas aproximada que se destinará a la resolución de cada paso. Los proyectos de referencia son de similar magnitud e igual ciclo de vida que el STS (Ver sección 3.i Proceso de desarrollo COMET).

Las salidas de cada iteración del proceso de desarrollo (prototipos incrementales entregables) están listadas en la siguiente tabla.

Entregables	Observaciones
Libthermalcore	Librería de cálculo térmico
libthermalserverapi	Librería que permite crear un servidor, que recibe pedidos de cálculo térmico
libthermalclientapi	Librería cliente para consultar datos al modelo térmico
Libstsmodels	Librería que modela componentes que modifican la temperatura del satélite y simula la evolución de la misma en el tiempo
Libstsapi	Librería que permite crear un servidor que recibe pedidos de la simulación térmica
Libstsclientapi	Librería cliente para consultar datos al simulador térmico
STSGUI	Interfaz gráfica del simulador térmico

33 Entregables del STS

En base a los entregables predefinidos y la designación de esfuerzo planificado, se definieron las tareas en un diagrama de Gantt con fecha de inicio el día 2 de Agosto de 2015 y de fin el día 14 de Enero de 2016. Este diagrama se encuentra disponible en el Apéndice E y muestra las dependencias entre las actividades y las tareas críticas.

Mientras se llevaba a cabo el STS, algunas de las estimaciones de esfuerzo se vieron modificadas. Las causantes de ello se explicaron en las Memorias Técnicas.

Como resultado de estas modificaciones, se replanificó el desarrollo y se obtuvo un mes y medio de retraso estimado. Esto queda evidenciado en el diagrama de Gantt representado en el Apéndice F actualizado durante las iteraciones del proceso de desarrollo y con fecha de fin el 22 de Marzo de 2016. Las tareas que presentaron cambios en el presupuesto de horas se encuentran marcadas con cursiva y * en el segundo diagrama de Gantt presente en el Apéndice y son las siguientes:

Tarea	% Aumento
12 – Capacitación	42
17 - Implementar las clases del modelo térmico	20
19 - Implementar el cálculo térmico	87
20 - Implementar ABM de relaciones entre nodos y relaciones de harness	42
23 - Establecer los métodos de la API libthermalserverapi	122
24 - Implementar servidor y registrar métodos	108
33 - Implementar servidor libstsapi que exponga la API y la de la libthermalcore	56

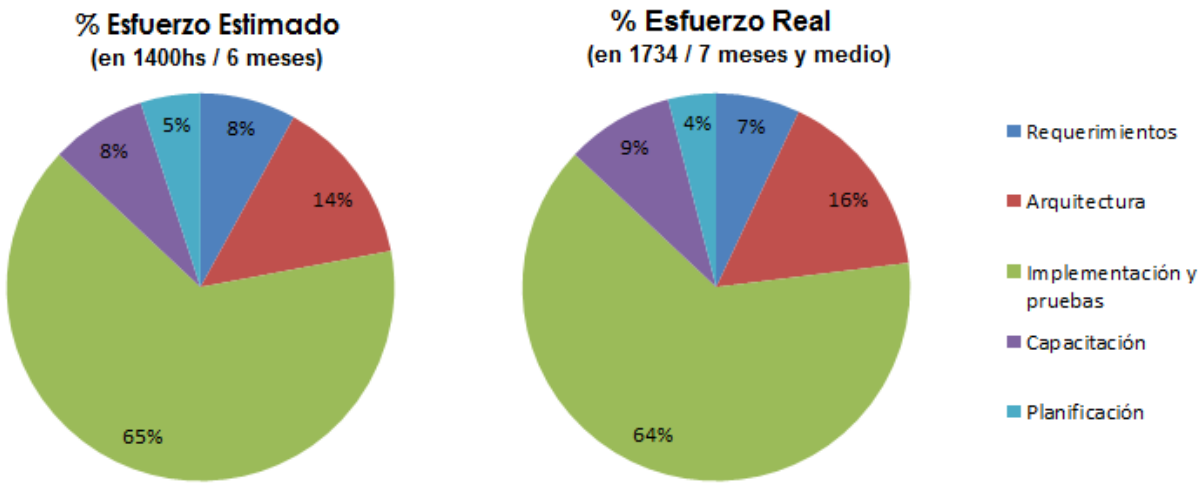
34 Reasignación de porcentaje de esfuerzo a las tareas del STS

Los porcentajes de esfuerzo finales separados por fase, quedaron del siguiente modo:

Actividades del Desarrollo de SW	Esfuerzo en horas	TOTAL %
Requerimientos	112	6,50%
Arquitectura	283	16,30%
Implementación y Pruebas	1110	64%
Capacitación	159	9,20%
Planificación	70	4%
TOTAL	**1734**	**100%**

35 Estimación de esfuerzo luego de iniciado el proyecto

Aquí una comparativa de los porcentajes estimados y los reales para tener en cuenta en siguientes proyectos:



36 Comparativa de las estimaciones realizadas

Se extrae como conclusión que es conveniente sobredimensionar las fases de mayor incertidumbre, que en este caso eran la arquitectura y la capacitación.

Respecto de la Arquitectura la incertidumbre dio como resultado, cambios en el diseño detallado de las distintas librerías.

Respecto de la Capacitación, la incertidumbre conllevó a una subestimación de la curva de aprendizaje y a errores, en algunos casos, de diseño y en otros de implementación.

iv. Avance de la tesina a distancia

El avance de la tesina a distancia transcurrió con normalidad. Es una buena opción para los estudiantes que con el tiempo dejamos de asistir a la Facultad.

La posibilidad de tener un director experto en los requerimientos de las tesinas y a su vez un asesor profesional que siga el avance día a día facilita la comunicación, el seguimiento del trabajo y de la investigación.

La posibilidad dada, de utilizar un proyecto laboral para llevarla a cabo, fue el puntapié para la recta final de la carrera, resolviendo gran parte del esfuerzo que requiere. Esta posibilidad sumó experiencia a mi desempeño laboral, ampliando mis conocimientos, capacidades técnicas, motivación y relaciones humanas.

El dominio espacial abarca múltiples tareas relacionadas con el desarrollo de SW y cada una de ellas con herramientas, restricciones y puntos de vista diferentes del problema. Participar de la creación de un simulador, implicó capacitarme en nuevos estándares, lenguajes de programación, herramientas de compilación, scripting y management, que se desarrollan en este documento.

La tesina fue redactada luego de la entrega del SW, e implicó volcar el conocimiento adquirido sustentado con las fuentes usadas para la inducción. Fue de suma importancia en este punto

contar con las memorias técnicas que se fueron completando a medida que se avanzaba en el desarrollo. Como así también, tener las tareas derivadas de los Casos de Uso, seleccionados para cada incremento funcional y las fuentes a las que se consultó. También fue útil disponer de la planificación inicial y el registro del tiempo de resolución real de las tareas para comparar y mejorar esta fase. En términos generales fue tan importante el relevamiento de contenido durante el desarrollo como el posterior procesamiento de la información que dio forma a esta tesina.

vii. Herramientas de Documentación

Durante el proceso de desarrollo del STS se utilizó como herramienta de documentación el Enterprise Arquitect (EA). Es una plataforma de interfaz intuitiva para diseño y modelado. En este caso se empleó como notación gráfica UML pero soporta otros estándares gráficos como por ejemplo SysUML. Fue de gran utilidad para concentrar los modelos en un único repositorio, de manera organizada. Entre las funcionalidades que provee, se aprovecharon las siguientes:

- Servicios Cloud

  Es un mecanismo de alojamiento de los modelos en servidores de la nube o de la empresa.

  Proporciona acceso fácil a personas de un mismo equipo. Esto fue una ventaja dado que los Stakeholders podían consultar los modelos o los requerimientos online con su usuario.

  También ofrece disponibilidad de los modelos en cualquier máquina con acceso a la red, lo cual se aprovechó en las primeras reuniones que se realizaron en distintos lugares físicos donde no se disponía de la PC de desarrollo.

- Reutilización de elementos UML

  Luego de definirse un componente UML con su descripción y relaciones en un diagrama puede ser necesario referenciarlo en otro gráfico. Para ello EA da la posibilidad de enlazarlo en el segundo diagrama de manera que toda la información relacionada a ese componente se concentre en una sola copia del mismo reduciendo el trabajo al modelar y el mantenimiento al momento de generar la documentación.

- Generación automática de documentos con plantillas

  Fue necesario, para este desarrollo, generar cierta documentación de respaldo para la empresa, con una estructura provista por una plantilla, carátula con datos particulares, etc. A medida que se iba modelando se completaron las descripciones y relaciones de cada componente de manera que la herramienta de generación de documentación, recopilara automáticamente estos datos y creara una primera versión de cada uno de los documentos solicitados. En los casos en que los modelos se modificaran solo hacía falta regenerar el documento para tenerlo actualizado.

- Trazabilidad de los requerimientos

  La trazabilidad es importante en los desarrollos de SW. Es significativo conocer el requerimiento que da origen al Caso de Uso, que se realiza en determinada clase y hasta si es posible, el commit en el repositorio con el código que lo implementa. La funcionalidad de trazabilidad que ofrece el EA se usó para relacionar Requerimientos con Casos de Uso, de manera que se pueda verificar la completitud del análisis. En la vista de Matriz de trazabilidad para cada Requerimiento se listan los CU que lo implementan. Esta matriz se muestra en la Especificación de Requerimientos del Apéndice A.

- Baselines

  EA ayuda a administrar la complejidad con herramientas de Baseline, permitiendo congelar el estado del proyecto por cada punto del tiempo y comparar dos versiones del mismo con la utilidad diff. Este mecanismo es útil al momento de agregar, modificar o eliminar requerimientos del sistema. De esta forma se puede seguir, analizar y comparar la evolución del análisis. También es posible verificar qué cambios se realizaron de una versión a otra del diagrama de clases o de entidad relación.

- Exportación de modelos

  Esta funcionalidad no es de las más usadas en el proceso de desarrollo. Sin embargo fue ventajosa al momento de pasar de un proyecto EA local al proyecto en el servidor ya que pudieron exportarse los modelos ya realizados a un formato XMI, sin necesidad de comenzar desde cero. De esta manera el archivo XMI puede importarse en un proyecto EA nuevo o en otra herramienta Case con soporte a este tipo de archivos

- Soporte al usuario

  Dado que fue el primer proyecto modelado con esta herramienta, hubo una etapa de aprendizaje. Se hizo uso del manual de usuario que está disponible en línea. El material se encuentra muy completo y es sencillo encontrar respuesta a las dudas que pueden surgir durante su uso. También Sparx recibe consultas de sus clientes y las resuelve rápidamente, como se pudo verificar al encontrarse un bug respecto de los tipos de datos en la exportación del diagrama de base de datos.

- Calendario del proyecto

  EA posee funcionalidad destinada a la administración del proyecto: se puede asignar recursos a las tareas, medir el esfuerzo y riesgos, estimar el tamaño y complejidad del proyecto, implementar control de cambios, realizar métricas de CU, etc. [RD27].

  En particular se hizo uso del calendario del proyecto para indicar hitos y sucesos acontecidos. Sin embargo los diagramas de Gantt utilizados en la planificación inicial de estos hitos, y las tareas a alto nivel fueron realizados con Microsoft Project, otra herramienta establecida en la empresa para planificación.

En líneas generales, el Enterprise Architect facilita la comunicación con Stakeholders, ayuda en la revisión de arquitectura, y es de gran utilidad para creación de documentos.

Se utilizaron los siguientes tipos de diagramas de la herramienta:

- Diagramas Estructurales: de Clases, Paquetes, Componentes y de Despliegue
- Diagramas de Comportamiento: Casos de Uso, Secuencia, de Estado y de Tiempo

En cuanto a la organización del proyecto EA, se corresponde con las etapas de COMET y sus distintas vistas del modelado. Por mencionar algunos:

- Vista de Casos de Uso:

En un paquete se detallan los diagramas de CU organizados por el nivel de detalle y la funcionalidad que describen. Además se detallan los requerimientos derivados de los mismos y se relacionan con los CU para generar la matriz de trazabilidad.

- Vista estática:

Representada en diagramas de clase UML tanto para el STS como para la LibThermalCore. Se relacionan las principales clases con los CU que implementan.

- Vista de implementación:

Esto representa una configuración específica de la arquitectura distribuida con componentes asignados a nodos de Hardware. Representado en diagramas de despliegue de UML. Se agrega además, la especificación de la base de datos para ambos casos.

- Vista de interacción dinámica:

Representada en diagramas de comunicación UML. En este caso utilizados para detallar Casos de Uso importantes.

- Vista dinámica de la máquina de estado:

Representada en diagrama de estados UML, utilizado para comprender los diferentes estados del simulador.

- Vista del componente estructural:

La arquitectura del STS se representó en términos de componentes, que están interconectados a través de interfaces.

Los documentos SRS y SDD se crearon de manera automática utilizando de base la estructura de paquetes mencionada.

De la experiencia del uso del EA se llega a la conclusión de que es una herramienta muy útil y amplia. Pudieron haberse explotado más las posibilidades que EA ofrece como la generación de código ejecutable, generación automática de scripts para creación de base de datos, seguridad basada en roles para ayudar a que las personas correctas contribuyan en la forma correcta, ingeniería inversa de las librerías heredadas para poder comprender más fácilmente su estructura estática, ingeniería inversa para actualizar los diagramas del desarrollo actual, etc. [RD27]. Además se utilizó otra herramienta para la gestión del proyecto, de haberse utilizado el EA se podría haber capturado y rastreado información de Testing, Project Management y mantenimiento, para tracear los cambios, conducir el desarrollo y la entrega del producto.

viii. Herramientas de Gestión

Planificación

En la etapa de planificación inicial como se detalló antes, se especificaron las tareas y los hitos o entregables a lo largo del tiempo presupuestado. Para ello desde la fecha inicial y según la estimación de la duración de las tareas, fue necesario organizarlas de manera que no se bloqueen unas con otras y que los recursos estén debidamente asignados. Esta organización de actividades se plasmó en un diagrama de Gantt que sirve de herramienta gráfica para visualizar las interdependencias entre ellas.

En el proyecto STS se utilizó para tal fin la herramienta Microsoft Project (MP). Este SW de administración de proyectos permite generar diagramas de Gantt, y así lograr el manejo de planes de trabajo, recursos, presupuestos y el análisis de cargas de trabajo. Ofrece plantillas de

proyectos para comenzar rápidamente y la posibilidad de realizar Baselines de comparación. Como desventaja, MP no es una herramienta online y solo es compatible con un sistema operativo.

Existen algunas herramientas que resuelven estas dos desventajas que presenta el MP entre ellas Project Online, Asana y Basecamp que son versiones basadas en la nube. Asana es libre para grupos de hasta 15 personas y Basecamp facilita la comunicación y colaboración del equipo.

Seguimiento de tareas

Para el seguimiento de tareas correspondientes a cada incremento se utilizó JIRA.

Es una aplicación, basada en el estándar J2EE, para la administración de proyectos y actividades desarrollada para facilitar el trabajo en equipo.

JIRA puede integrase con otras herramientas de Attlasian como Bitbucket o Stash para lograr trazabilidad de líneas de código con tareas, también puede integrarse con Crucible para realizar revisiones de código. Existe también la herramienta Portfolio for JIRA que permite llevar a cabo planificaciones y tomar decisiones con el respaldo de los datos cuando se produzcan imprevistos y tener a todo el equipo al tanto.

Puede instalarse JIRA en la nube, en un servidor propio o un data center.

Desde el punto de vista de administración de tareas del STS, JIRA aportó los siguientes beneficios:

- Simplicidad, su interfaz es amigable y fácil de aprender
- Administración de diferentes tipos de tareas, errores, historias, épicas.
- Permite manejar la lista de tareas del backlog realizando un ranking, priorizando y planificando el trabajo.
- Además permite aplicar filtros rápidos configurables. Algunos por defecto son: filtro por prioridad, solo mis tareas y tareas actualizadas recientemente.
- Notificaciones vía email.
- Fácil extensión e integración con otros sistemas.
- El panel de control permite ver toda la información relevante para el usuario en un vistazo
- Provee la posibilidad de asignar diferentes permisos a usuario, se puede configurar el flujo de trabajo, y además cada usuario puede configurar su panel de control de acuerdo a sus necesidades, poniendo la información clave a su disposición.
- Se pueden crear subtareas de una historia para partirla en funciones implementables.
- Permite agrupar las historias relacionadas en épicas de manera que puedan filtrarse las tareas rápidamente.

Algunas funcionalidades que ofrece JIRA enumeradas en [RD53] que no se utilizaron en este proyecto:

- Creación de informes
- Para organizar los sprints existe un marcador que permite ver cuánto trabajo se puede asignar a un sprint en particular, valuado en Story Points, horas, etc. El marcador aparece como una barra azul en medio del backlog. El modo de uso es estimar las tareas en Story Points, luego realizar un seguimiento de las horas de trabajo que costó cada tarea y con ello retroalimentar las métricas de estimación, de manera que el marcador indique cuantas

tareas implican el esfuerzo usualmente asignado a un sprint para asegurar que se está planificando de acuerdo al rendimiento del equipo de trabajo. Así se puede conocer la velocidad de trabajo del equipo y utilizar este dato para futuras estimaciones.

Como desventaja se puede mencionar que tanto JIRA como los plugins de extensión tienen licencia paga.

En base a la experiencia, realmente la interfaz de la organización de tareas es muy amigable y sencilla. Se utilizó instalada en un servidor propio, con plugins para trazabilidad con GIT.

Seguimiento de errores

Achievo es una herramienta web de apoyo a los procesos de negocio que ofrece seguimiento de proyectos, clientes, contactos, planificación y programación diaria. Es de código abierto, independiente de la plataforma y se construye a partir de varios módulos y una base de datos central.

Esta herramienta se encuentra implementada con un framework web llamado ATK, que está codificado en PHP, cuyo propósito son las aplicaciones empresariales.

Achievo en su manual [RD54] ofrece una amplia variedad de funcionalidades entre las que se destacan:

- Manejo de varios proyectos, sus estados y contactos
- Planificación a lo largo del proyecto, diagramas Gantt y estadísticas
- Plantillas de proyectos
- Empleados y roles
- Citas, calendarios compartidos y registro de horas usadas en el proyecto
- Reportes
- Vista de actividades diaria, semanal, mensual
- Notas y TODO’s del proyecto

En el proyecto de esta tesina, la herramienta fue utilizada para el reporte y seguimiento de bugs. Allí se definió el proyecto STS con la información administrativa entre las que se encuentran el presupuesto, el solicitante, el responsable del producto y fechas de inicio y entrega.

Es posible asignar tickets al proyecto como TODO’s, donde se informa el error, el procedimiento para reproducirlo, en qué versión se halló, su prioridad, quién lo reporta y a quien lo asigna. De esta manera se notifica al desarrollador el nuevo bug pendiente; luego de que este es analizado puede notificarse como en progreso o resuelto y puede ser verificado y cerrado por el solicitante.

Capítulo 5: Instalación del STS

i. Ambiente de desarrollo

Este proyecto se desarrolló mayormente en un entorno laboral. Pero dado que era necesaria la disponibilidad del ambiente de desarrollo y de ejecución para poder inspeccionar el código, probarlo y consultarlo para su posterior demostración en la defensa de la tesina, se optó por un ambiente virtualizado que permita conseguir la portabilidad suficiente para tal motivo.

Tanto el STS como la LibThermalCore, por requerimiento, se desarrollaron y serán usadas en producción en un ambiente Ubuntu 12.04 de 64 bits. La herramienta de virtualización seleccionada fue VBOX 4.3.30 en la que se creó una máquina virtual con el Sistema Operativo requerido.

En el proceso de creación de la máquina virtual se optó por una imagen x86_64/AMD para que sea de 64 bits como era requerido, esto es importante por la compatibilidad con las herramientas de desarrollo disponibles y las librerías usadas como dependencias. Para los siguientes pasos de la configuración fue necesario disponer de un usuario con privilegios y grupo root. A continuación se instalaron las Guest Aditions provistas por la VM.

Se creó una carpeta compartida con la maquina host, se configuró la red en la VM para que utilice el proxy de la empresa según se explica en [RD55] y se configuró el proxy en los navegadores.

El siguiente paso fue instalar la herramienta de versionamiento GIT y la herramienta gráfica GITK para tener un seguimiento más amigable de las versiones del código. Para su instalación debe usarse apt-get un gestor de paquetes instalables. Uno de los problemas frecuentes al usar apt-get es que se encuentran lockeadas las carpetas que utiliza, el problema y la solución se detallan en [RD56].

Se instaló el openjdk-7-jdk con el gestor de paquetes, antes de descargar el IDE de desarrollo Eclipse Juno para 64 bits.

Luego se clonaron los proyectos de librerías existentes para conocer su código y se crearon los proyectos de librerías STS y LibThermalCore en el repositorio remoto.

ii. Instalación del STS y Puesta en Marcha

Para la instalación de la herramienta STS es necesario tener acceso a los repositorios de las librerías del STS y sus dependencias:

Repositorio	Dependencias
Librerías Servidor	
ThermalCore	libglog, libdssdata, libsmp2
STS	libsmp2 , libdssapi, libthermalserverapi, libthermalcore
Librerías Cliente	
libthermalcoreclientapi	libdsscommon, libdssapi
libstsclientapi	libdsscommon, libdssapi
STSGUI	PyQt4

37 Repositorios del STS

Acceder al directorio de trabajo y a través del comando:

git clone *http://urlDelRepositorio*

Descargar todos los proyectos especificados en la anterior tabla.

Compilar con Autotools, en el siguiente orden los proyectos:

- ../Thermalcore/lib/libthermalcore
- ../Thermalcore/lib/libthermalserverapi
- ../STS/lib/libstsmodels
- ../STS/lib/libstsapi
- ../libthermalcoreclientapi
- ../libstsclientapi

Esto significa que en la carpeta donde se encuentra el Makefile ejecutar:

./autogen.sh && ./configure && make && sudo make install && sudo ldconfig

Compilar el proyecto STSGUI con el comando make.

El STS posee un archivo de variables de configuración en el que se debe actualizar la información indicando las rutas correspondientes a la base de datos térmica y del simulador.

El modo de poner en marcha tanto el servidor como la interfaz gráfica ejecutando el script ./run que contiene cada uno en su carpeta /src.

iii. Software User Manual (SUM)

Tipos de lectores

Este documento contempla la interfaz gráfica del simulador y está dirigido a los usuarios finales del sistema, quienes se espera tengan conocimientos respecto de modelos térmicos y su configuración.

Versiones aplicables al manual

Este manual aplica a la versión 1.0 del STS.

Propósito

El objetivo de este documento es describir el manual de usuario de las interfaces gráficas del Simulador Térmico Satelital (STS).

Contenido del documento

En las siguientes secciones se describe la funcionalidad y operación del STS a través de su interfaz gráfica.

Descripción general

El Simulador Térmico Satelital reproduce el comportamiento del modelo térmico reducido planteado para representar la propagación de la temperatura a través de sus componentes.

Tiene como propósito:

- Validar el modelo térmico del satélite.
- Usarse como herramienta de diseño durante la creación del modelo.
- Usarse como herramienta de verificación para reproducir casos particulares de variaciones de temperatura.

Funciones

Las funciones con las que cuenta el STS, son:

- Implementación de Simulador y servicios de Scheduler, TimeKeeper, Logger, EventManager.
- Modelo Térmico: refactoring de la infraestructura del modelo térmico (kernel de cálculo de ecuaciones de temperaturas) e infraestructura de modelos de harness térmico.
- Tablas de BD asociadas al modelo térmico y componentes de harness térmico.
- Interfaz que permite la comunicación con la aplicación de interfaz gráfica de usuario (GUI) y brindar acceso a la Suite de Pruebas con el objetivo de verificar el correcto funcionamiento del modelo que lo compone.
- Interfaz que permite el control de la simulación del STS.

Requerimientos Mínimos de Hardware y Software

PC compatibles Intel I7 con 8 GB de Memoria RAM y HDD de 500 GB.

Recomendado: Server Intel Xeon con 12 cores, 16 Gb de RAM y HDD de 500 GB

Sistema Operativo Ubuntu v.12.04 actualizado, 64 bits.

Placa de Red de 100 Mbps.

Monitor LCD 20''

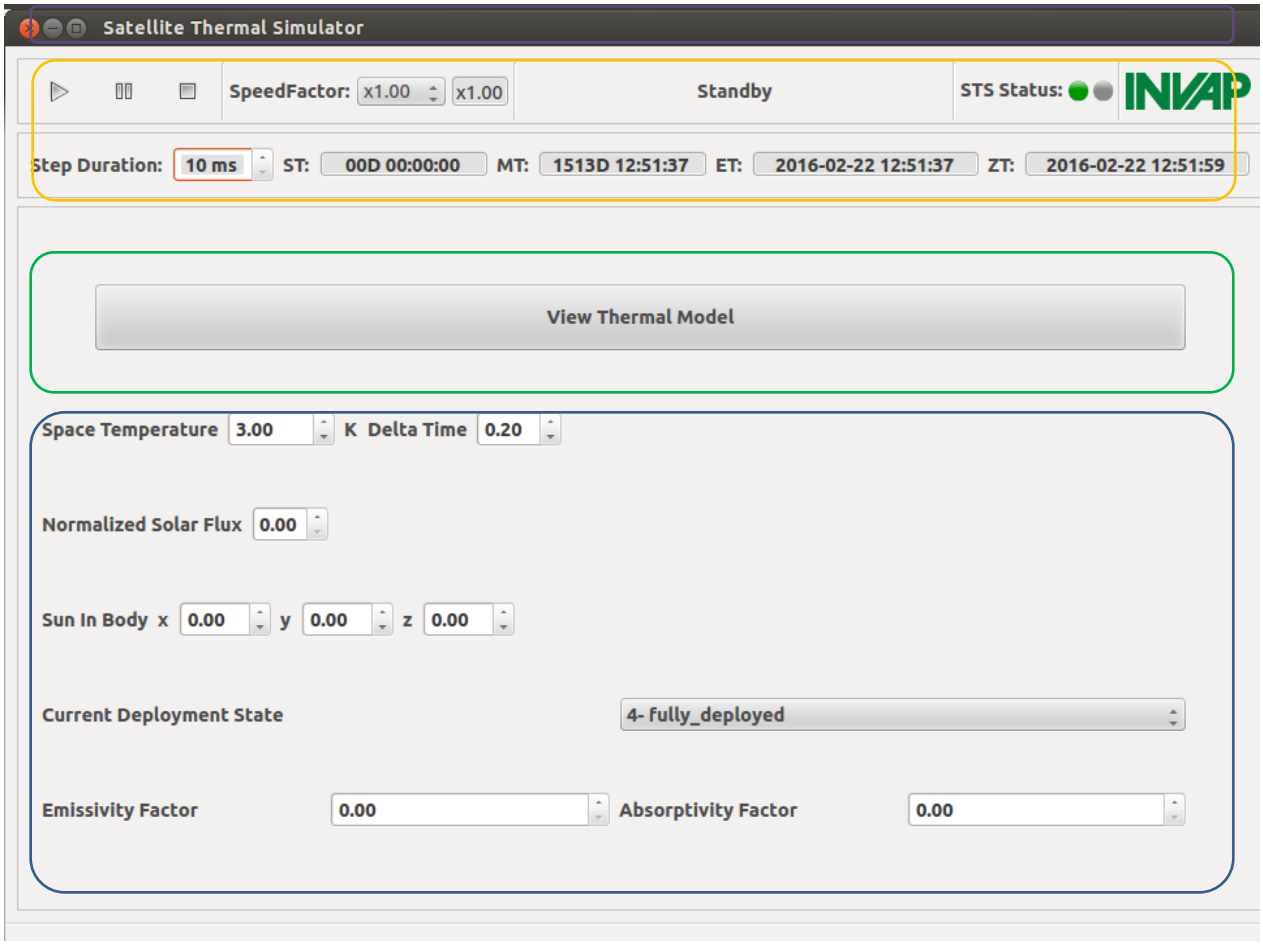
Operaciones a través de la interfaz gráfica STSGUI

Desde la pantalla principal se tiene acceso a un menú desde el cual se puede acceder a los controles de simulación, las preferencias del sistema relacionadas a la conexión y a la sección de información térmica. Además cuenta con un sector de configuración del entorno en el que se simula el modelo térmico.

Ventana Principal

En esta ventana aparecen los controles y configuraciones de la simulación, motivo por el cual, todos los campos se encontrarán deshabilitados hasta que exista una conexión con el servidor del simulador. Por medio de los leds STS Status puede observarse en rojo que el servidor está desconectado y verde conexión habilitada. Para lograr esta conexión debe lanzarse el simulador

como se explicó previamente en la sección Puesta en Marcha del STS. Esta conexión habilitará los controles.



38 Ventana principal del STS

Botón de acceso al modelo térmico

Por medio de este botón se accede a la ventana de visualización del modelo térmico que permite el seguimiento de nodos, componentes sensores, componentes disipadores, superficies y relaciones entre nodos.

Sector de Configuración de Entorno

Esta sección permite brindar parámetros que representan la incidencia del Sol en el modelo, la posición de despliegue del satélite, y aquellos relacionados al cálculo de la ecuación térmica, estos datos modificarán los resultados obtenidos.

- Normalized Solar Flux

Este valor puede variar de [0..1] e indica el flujo del sol normalizado. Se usa en el cálculo de las zonas de eclipse de las caras del satélite.

- Space Temperature

Este valor está expresado en grados Kelvin. Siendo 3 grados por defecto. Indica la temperatura con la que se estima contará el espacio durante el cálculo térmico, dado que la temperatura se pierde hacia el espacio.

- Delta Time

Se define como delta time al tiempo de avance en un paso del cálculo térmico. El deltaTime es el tamaño en segundos que dura cada paso de la simulación. El rango de valores para el parámetro puede variar de [0,05..0,5].

Por defecto este valor es 0,2 seg. Este valor puede modificarse con el simulador en estado Standby.

- Sun In Body(x y z)

Este parámetro representa la posición del sol respecto del satélite, ya que el sol modificará la temperatura a la que los nodos están sometidos. La configuración del sol está definida como vector en alguna de las diferentes posiciones de la esfera que contiene al modelo.

Por defecto no se tiene en cuenta la presencia del sol.

- Deployment State

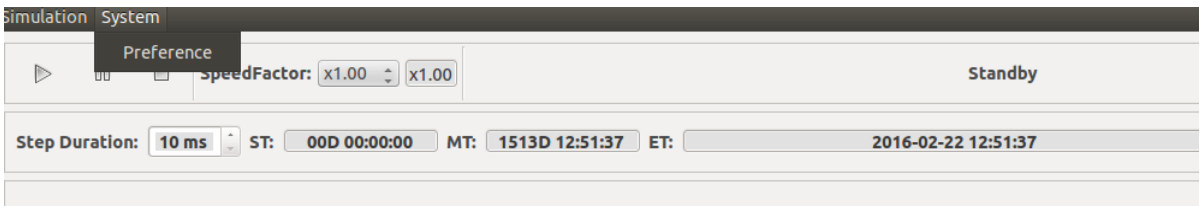
El estado de despliegue hace referencia a la posición geométrica de las superficies del satélite. Por lo general existe un estado GEO para representar la posición final de las partes mecánicas (antena, paneles solares) cuando el satélite esta en órbita; GTO es la disposición de las partes cuando el satélite está separando o desplegando sus partes móviles, y STOWED que representa las piezas plegadas sobre el módulo central que es la ubicación de las superficies en el lanzamiento. Sin embargo cada modelo implementará las propias y serán listadas en este combo, pudiéndose seleccionar una en particular para la simulación.

- Emissivity & Absorptivity Factors

Estos factores son utilizados en el cálculo para representar a modo porcentual el grado con el que el satélite refracta o absorbe los rayos del Sol. Pueden variar de [0..1]. Siendo por defecto 0 para ambos casos.

Menú Principal

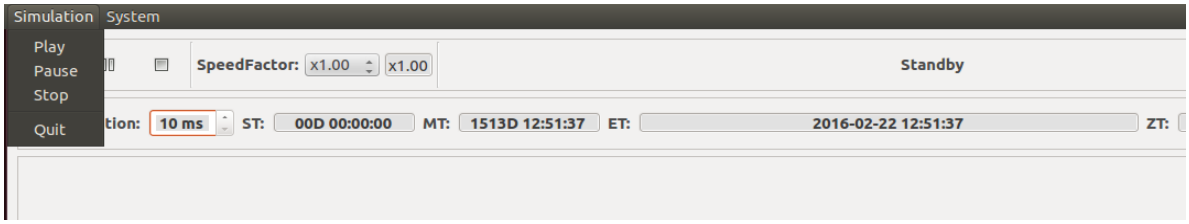
Este menú ubicado en la zona superior de la ventana provee las siguientes opciones:



39 Menu principal

- Preferencias

Este submenú contiene la dirección “IP Address” donde corre el STS y puerto “TCP Port” que puede ser cualquier puerto TCP valido (>1024), se recomienda 8890



40 Preferencias

- Simulation

En el Submenú “Simulation” se puede optar por salir de la aplicación con la opción “Quit” o comandar el estado de la simulación con “Play”, “Pause”, “Stop”.

Control de Simulación

- State Label

Se encuentra ubicado en el medio del sector superior, muestra el estado actual del STS con la leyenda ‘Executing’ o ‘Standby’.

- Simulation State Controls: Play

Para correr la simulación, se debe seleccionar “play” esto se puede hacer a través del menú

“Simulation->Play” o utilizando el botón que contiene el icono de play (“►”) .

En este caso el simulador STS comienza a ejecutarse, se puede confirmar este estado observando la leyenda “Executing”.

- Simulation State Controls: Pause

Para pausar la simulación, se debe seleccionar “Pause” a través del menú

“Simulation->Pause” o el botón pause (“▌▌”).

El simulador quedará en pausa y se observará la leyenda en estado de “Standby”.

- Simulation State Controls: Stop

Para detener la simulación, se selecciona “Stop” a través del menú “Simulation ->Stop” o utilizando el botón que representa el ícono de stop (“■”).

Se observará también para este caso el estado de “Standby” pero los tiempos se reiniciarán.

- SpeedFactor

Es la velocidad deseada a la cual debe correr la simulación, se observa en el sector superior de la ventana principal con la denominación de “SpeedFactor”, este valor puede cambiarse en cualquier momento de la simulación utilizando el “spin box” que por defecto está en x1.00.

- Running Factor

Es la velocidad a la que realmente corre la simulación, este se encuentra a continuación del “spin box” del speed factor en un label griseado.

- Connection Enable

Representación de la conexión de la GUI con el servidor STS mediante dos “leds” ubicados en el extremo superior derecho (verde: conectado; rojo: no conectado).

- Step Duration

Es el tiempo de paso de la simulación, por defecto este tiempo es de 25ms. y puede ser modificado en el spin box que lo contiene.

- Simulation Time

Tiempo que lleva corriendo la simulación, comienza en cero y se incrementa en relación a la velocidad con la que corre la simulación. Cuando se pausa la simulación este valor es reiniciado, su valor se observa en “ST”.

- Mission Time

Es el tiempo de la misión, su valor se observa en “MT”.

- Epoch Time

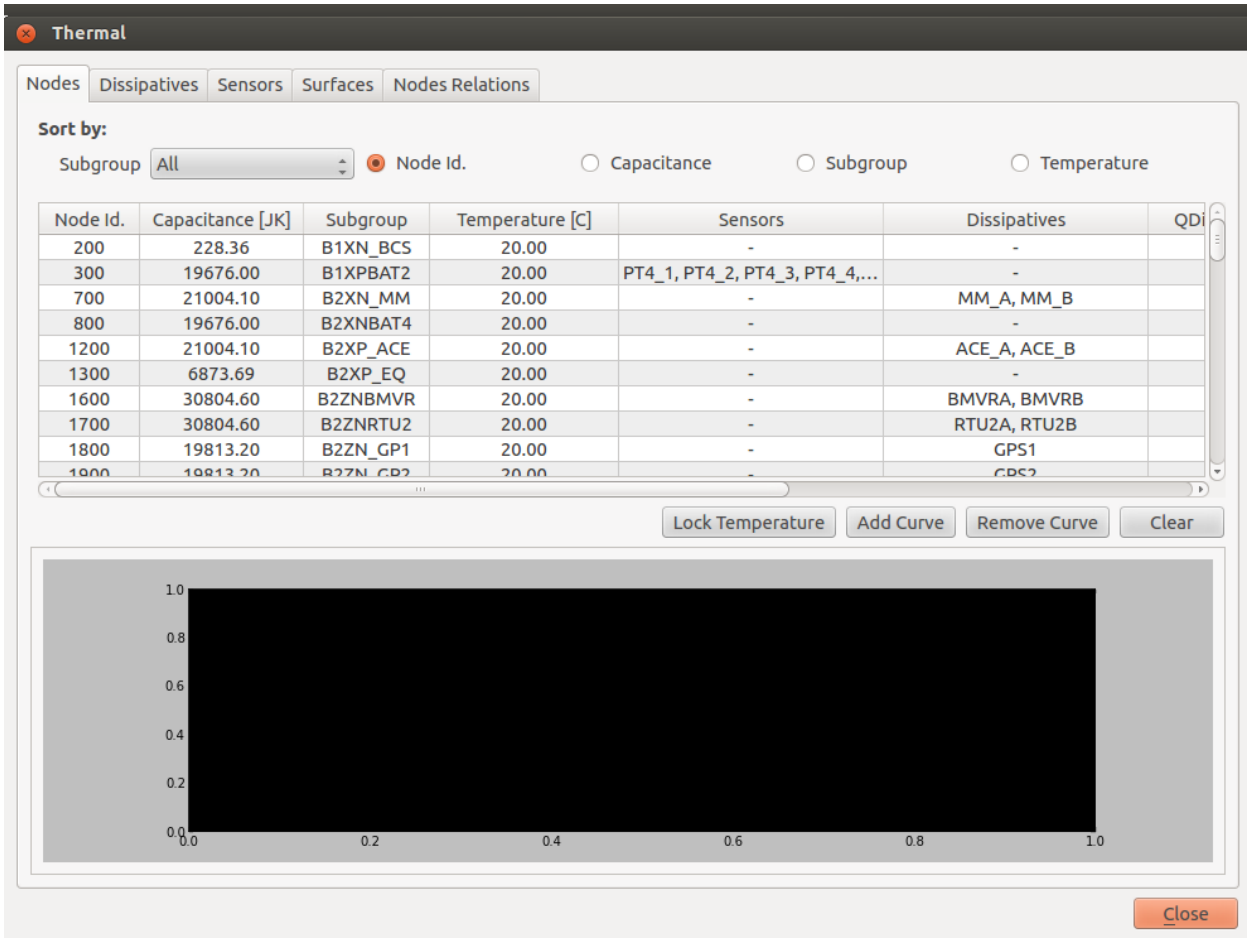
Hace referencia a la fecha de simulación, su valor se observa en “ET”.

- Zulu Time

Es la fecha actual del servidor sobre el cual corre el STS. Su valor se observa en “ZT”.

Herramienta de Inspección del Modelo Térmico: pestaña de nodos

En la pantalla inicial de la herramienta de inspección del modelo térmico puede observarse el conjunto de todos los nodos que forman parte del Modelo Reducido Térmico del STS.



41 Pestaña de inspección de nodos

Para cada nodo se especifica una serie de valores de interés a la hora de verificar el funcionamiento del mismo. En primer lugar, se muestra el número que identifica al nodo, la masa y la capacidad calorífica del mismo, el subgrupo del que forma parte, de esta manera los nodos se pueden agrupar; y la temperatura instantánea del nodo.

Existen también dos columnas donde se listan los componentes relacionados al nodo: aquellos que se encargan de censar el entorno, aquellos que disipan potencia y las superficies de las que forman parte. Estas relaciones se conocen como harness térmico de sensores, disipadores y superficies respectivamente. Para cada nodo además, se muestra la sumatoria de la disipación que generan los disipadores que tiene asociados.

Se denominan sensores a Termistores y Termostatos.

Se denominan disipadores a las Cargas (Loads), ya sean de Resistencia Continua o Resistencia y Potencia Continuas, la batería, capacitor de regulación, heaters y cualquier otro componente que disipe temperatura.

La información de estos componentes se extrae de la base de datos del STS.

- Filtrado y orden de Nodos

En el sector superior izquierdo se encuentra un combo con el que se pueden filtrar los nodos a inspeccionar según el subgrupo al que pertenecen. Por otro lado con la opción “All” se listan todos los nodos a la vez.

Hacia el sector derecho se dispone de opciones para ver los nodos ordenados según el criterio elegido: NodeId, Capacitancia, Subgrupo o Temperatura instantánea.

- Locked Temperature

Debajo de la tabla de nodos puede encontrarse la opción “Lock Temperature”, que permite bloquear la temperatura de determinado nodo una vez que alcance el monto especificado.

Asimismo se reserva la última columna para indicar la temperatura aplicada al lockear un nodo. Esto permite chequear el valor con el que fue lockeado mientras el cálculo térmico hace tender la temperatura real del nodo a la temperatura elegida.

- Seguimiento gráfico

Otra funcionalidad que permite esta vista es hacer el seguimiento gráfico de algunos nodos en particular. Para ello, se deben seleccionar de la tabla cada nodo, y presionar el botón “Add Curve”, para que comience a graficarse la variación de la temperatura a través del tiempo.

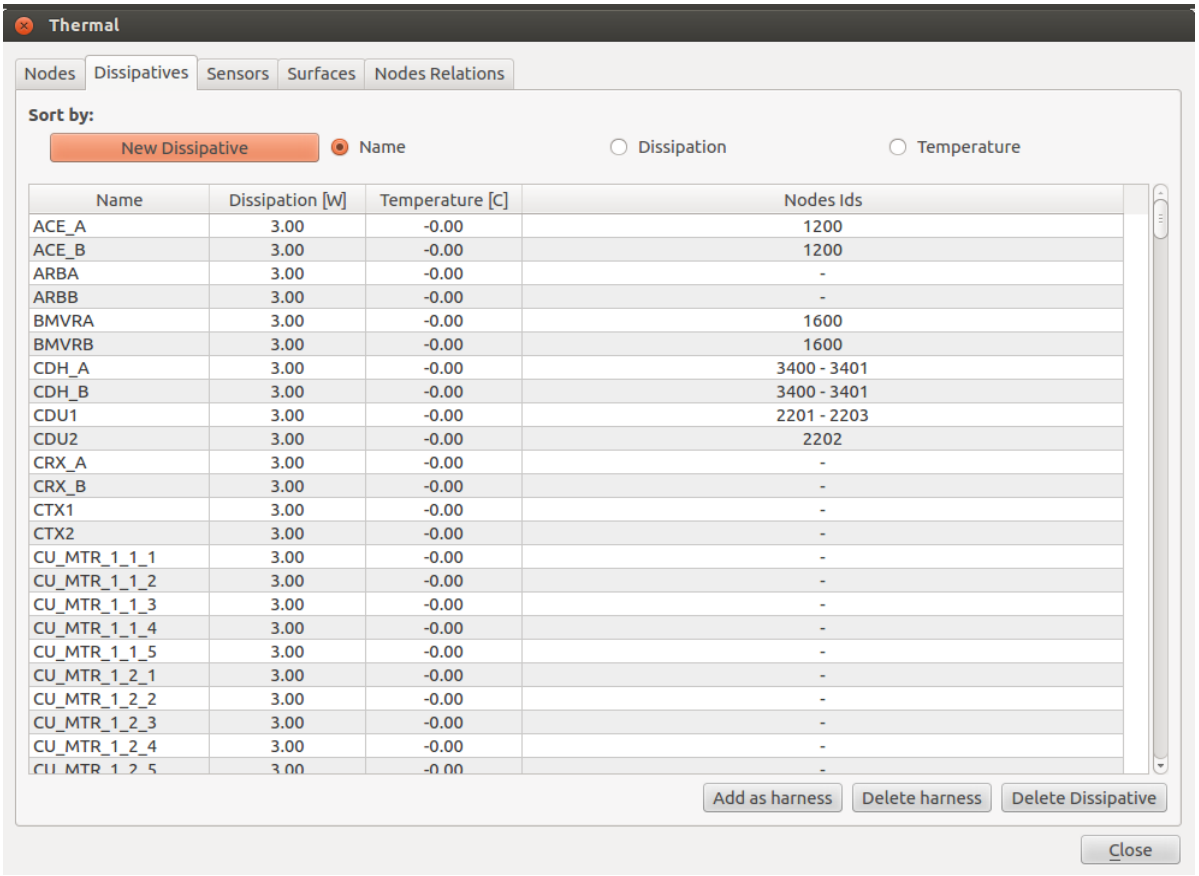
El botón “Remove Curve” se utiliza para quitar alguno de los nodos graficados y el botón “Clear” limpia todas las series del gráfico.

Modificaciones volátiles del Modelo

En las pestañas a continuación pueden realizarse modificaciones al modelo. Estas se llaman “volátiles” dado que no se almacenan en la base de datos, permiten realizar pruebas para intentar diferentes mejorías al modelo térmico, pero no serán persistidas de una ejecución del servidor a otra.

Pestaña de Disipadores Térmicos

Todos los Disipadores se visualizan en la solapa “Dissipatives” de la ventana de inspección del modelo.



42 Pestaña de inspección de disipadores

De cada disipador del modelo se puede obtener el nombre, la disipación que genera, la temperatura y los nodos con los que se relaciona (Harness).

- Filtrado y orden de Disipadores

Se los puede ordenar por Nombre, Disipación o Temperatura eligiendo el “radio button” correspondiente.

- Modificaciones de disipadores en el Modelo

Entre las opciones posibles se encuentra, en el sector superior izquierdo, el botón “Add Dissipative” que permite en una nueva ventana, asignar un nombre a un nuevo componente que disipe cierta cantidad de watts, con determinada temperatura. La nueva instancia se listará en la tabla junto a los demás.

El SW no valida si el nombre del nuevo disipador ya está en uso, se espera que los nombres sean usados como clave identificadora de los disipadores.

Debajo de la tabla de disipadores se encuentran las opciones para realizar modificaciones al harness.

- Add as harness

Permite relacionar el disipador seleccionado en la tabla con alguno de los nodos del modelo. Para ello, debe seleccionarse la fila del disipador correspondiente, presionar el botón “Add as harness”, que abrirá una ventana de diálogo nueva con un combo que lista los id’s de los nodos disponibles, seleccionar uno y luego presionar “Ok”.

La interfaz gráfica valida, la duplicidad de las relaciones, es decir que no podrán existir dos relaciones de harness entre un mismo disipador y un nodo. Sin embargo la librería térmica no lo valida podrían existir dos relaciones entre el mismo nodo y componente, el usuario de la LibThermalCore deberá evaluar si es correcto o no.

- Delete harness

Permite borrar la relación entre el disipador seleccionado en la tabla con alguno de los nodos que muestra en la columna de Nodes Ids. Para ello, debe seleccionarse la fila del disipador correspondiente, presionar el botón “Delete harness”, que abrirá una ventana de diálogo nueva con un combo que lista los id’s de los nodos con los que se encuentra relacionado, seleccionar uno y luego presionar “Ok”.

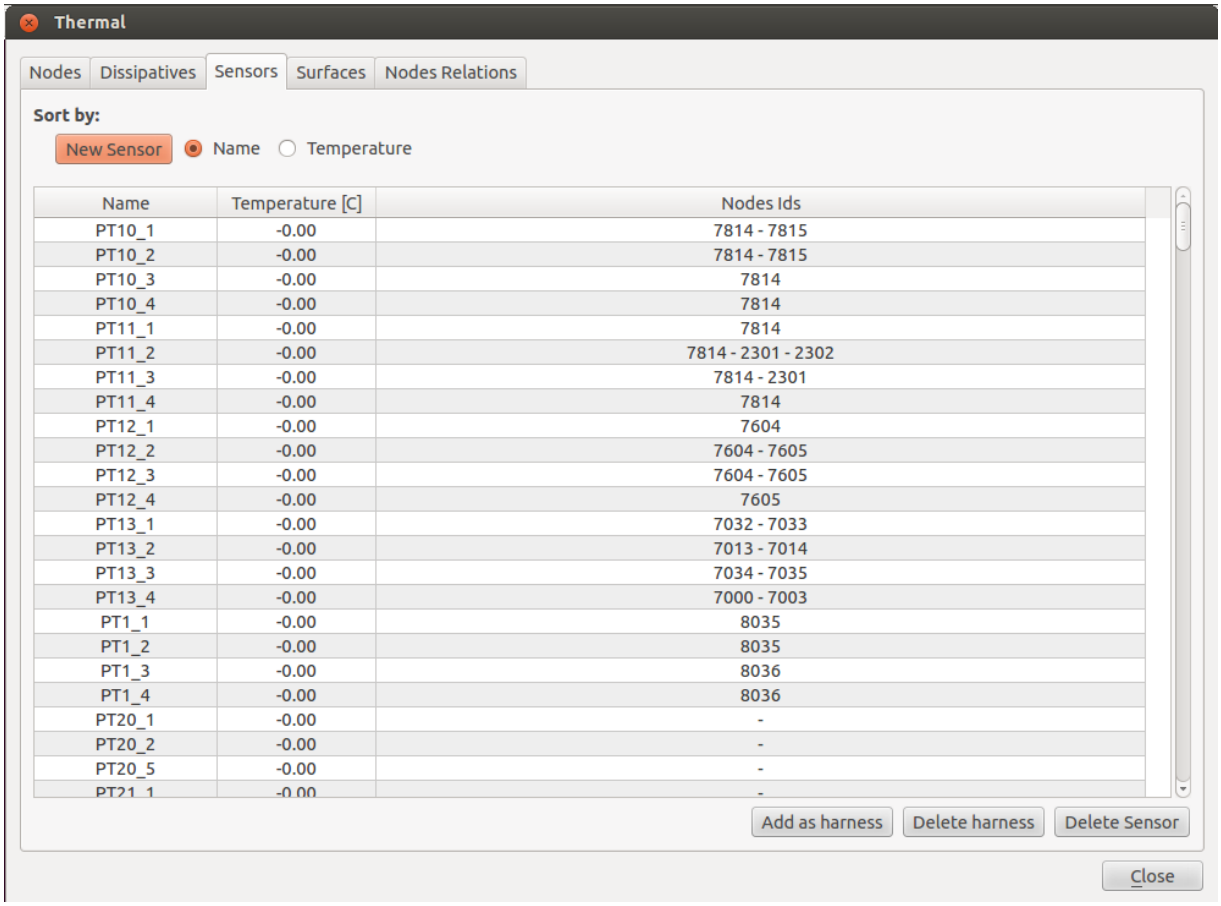
- Delete Dissipative

Permite borrar el componente disipador seleccionado en la tabla. Para ello, debe seleccionarse la fila del disipador correspondiente, presionar el botón “Delete Dissipative”, confirmar la ventana de diálogo y el disipador se removerá de la lista.

El SW valida si el disipador está relacionado con algún nodo, en cuyo caso primero el usuario deberá borrar la relación y después el componente.

Pestaña de Sensores Térmicos

Todos los Sensores se visualizan en la solapa “Sensors” de la ventana de inspección del modelo.



Name	Temperature [C]	Nodes Ids
PT10_1	-0.00	7814 - 7815
PT10_2	-0.00	7814 - 7815
PT10_3	-0.00	7814
PT10_4	-0.00	7814
PT11_1	-0.00	7814
PT11_2	-0.00	7814 - 2301 - 2302
PT11_3	-0.00	7814 - 2301
PT11_4	-0.00	7814
PT12_1	-0.00	7604
PT12_2	-0.00	7604 - 7605
PT12_3	-0.00	7604 - 7605
PT12_4	-0.00	7605
PT13_1	-0.00	7032 - 7033
PT13_2	-0.00	7013 - 7014
PT13_3	-0.00	7034 - 7035
PT13_4	-0.00	7000 - 7003
PT1_1	-0.00	8035
PT1_2	-0.00	8035
PT1_3	-0.00	8036
PT1_4	-0.00	8036
PT20_1	-0.00	-
PT20_2	-0.00	-
PT20_5	-0.00	-
PT21_1	-0.00	-

43 Pestaña de inspección de sensores

De cada sensor se puede obtener el nombre, la temperatura y los nodos con los que se relaciona (Harness).

- Filtrado y orden de Sensores

Se los puede ordenar por Nombre o Temperatura eligiendo el “radio button” correspondiente.

- Modificaciones de sensores en el Modelo

Entre las opciones posibles se encuentra, en el sector superior izquierdo, el botón “Add Sensor” que permite en una nueva ventana, asignar un nombre a un nuevo componente con determinada temperatura, el nuevo sensor se listará en la tabla junto a los demás.

El SW no valida si el nombre del nuevo sensor ya está en uso, se espera que los nombres sean usados como clave identificadora de los sensores.

Debajo de la tabla de sensores se encuentran las opciones para realizar modificaciones al harness.

- Add as harness

Permite relacionar el sensor seleccionado en la tabla con alguno de los nodos del modelo. Para ello, debe seleccionarse la fila del sensor correspondiente, presionar el botón “Add as harness”, que abrirá una ventana de diálogo nueva con un combo que lista los id’s de los nodos disponibles, seleccionar uno y luego presionar “Ok”.

La interfaz gráfica valida, la duplicidad de las relaciones, es decir que no podrán existir *dos relaciones de harness entre un mismo disipador y un nodo. Sin embargo la librería* térmica no lo valida podrían existir dos relaciones entre el mismo nodo y componente, el usuario de la LibThermalCore deberá evaluar si es correcto o no.

- Delete harness

Permite borrar la relación entre el sensor seleccionado en la tabla con alguno de los nodos que muestra en la columna de Nodes Ids. Para ello, debe seleccionarse la fila del sensor correspondiente, presionar el botón “Delete harness”, que abrirá una ventana de diálogo nueva con un combo que lista los id’s de los nodos con los que se encuentra relacionado, seleccionar uno y luego presionar “Ok”.

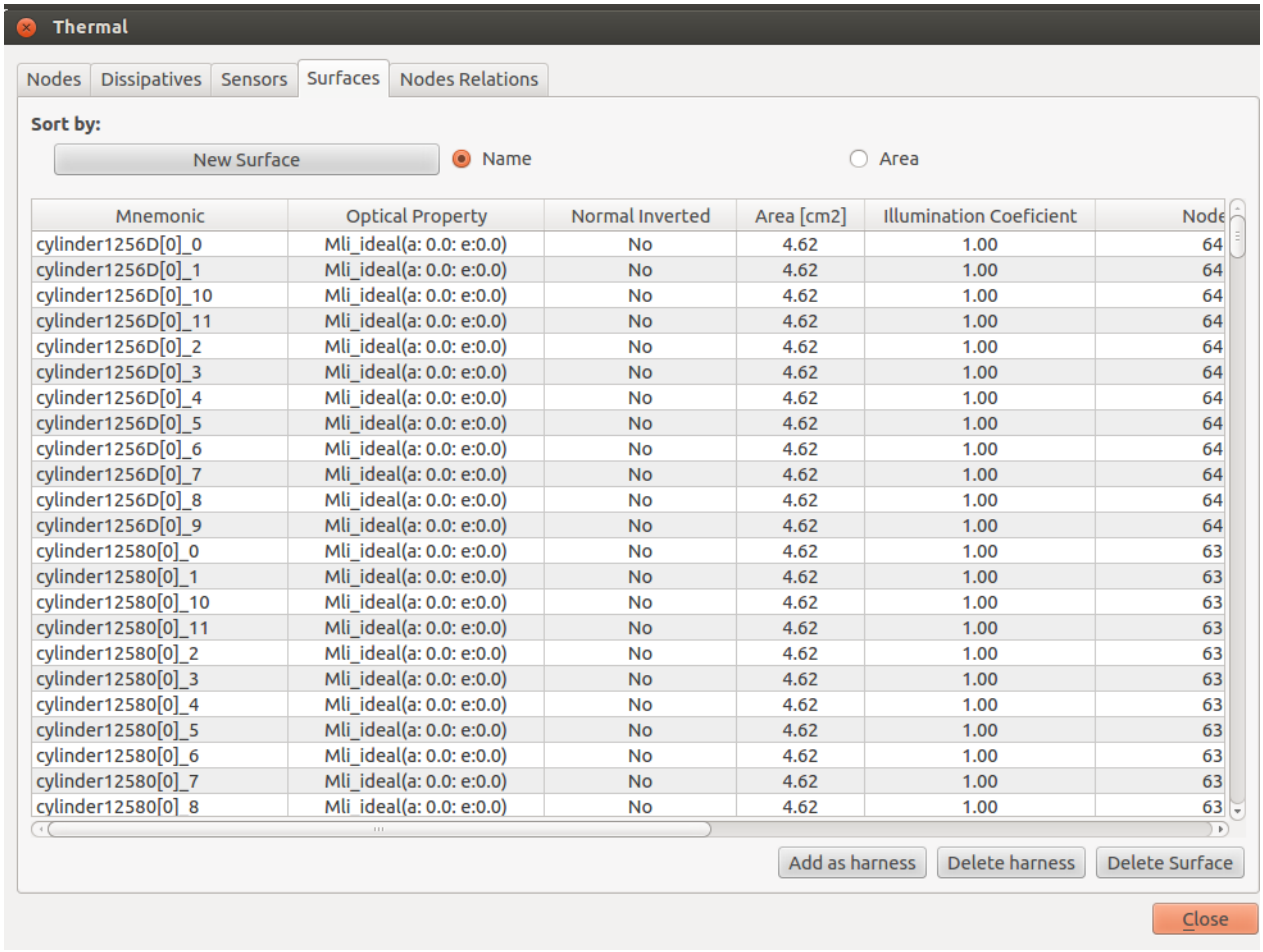
- Delete Sensor

Permite borrar el componente sensor seleccionado en la tabla. Para ello, debe seleccionarse la fila del sensor correspondiente, presionar el botón “Delete Sensor”, confirmar la ventana de diálogo y el sensor se removerá de la lista.

El SW valida si el sensor está relacionado con algún nodo, en cuyo caso primero el usuario deberá borrar la relación y después el componente.

Pestaña de Superficies

Todas las Superficies se visualizan en la solapa “Surfaces” de la ventana de inspección del modelo.



Mnemonic	Optical Property	Normal Inverted	Area [cm2]	Illumination Coeficient	Node
cylinder1256D[0]_0	Mli_ideal(a: 0.0: e:0.0)	No	4.62	1.00	64
cylinder1256D[0]_1	Mli_ideal(a: 0.0: e:0.0)	No	4.62	1.00	64
cylinder1256D[0]_10	Mli_ideal(a: 0.0: e:0.0)	No	4.62	1.00	64
cylinder1256D[0]_11	Mli_ideal(a: 0.0: e:0.0)	No	4.62	1.00	64
cylinder1256D[0]_2	Mli_ideal(a: 0.0: e:0.0)	No	4.62	1.00	64
cylinder1256D[0]_3	Mli_ideal(a: 0.0: e:0.0)	No	4.62	1.00	64
cylinder1256D[0]_4	Mli_ideal(a: 0.0: e:0.0)	No	4.62	1.00	64
cylinder1256D[0]_5	Mli_ideal(a: 0.0: e:0.0)	No	4.62	1.00	64
cylinder1256D[0]_6	Mli_ideal(a: 0.0: e:0.0)	No	4.62	1.00	64
cylinder1256D[0]_7	Mli_ideal(a: 0.0: e:0.0)	No	4.62	1.00	64
cylinder1256D[0]_8	Mli_ideal(a: 0.0: e:0.0)	No	4.62	1.00	64
cylinder1256D[0]_9	Mli_ideal(a: 0.0: e:0.0)	No	4.62	1.00	64
cylinder12580[0]_0	Mli_ideal(a: 0.0: e:0.0)	No	4.62	1.00	63
cylinder12580[0]_1	Mli_ideal(a: 0.0: e:0.0)	No	4.62	1.00	63
cylinder12580[0]_10	Mli_ideal(a: 0.0: e:0.0)	No	4.62	1.00	63
cylinder12580[0]_11	Mli_ideal(a: 0.0: e:0.0)	No	4.62	1.00	63
cylinder12580[0]_2	Mli_ideal(a: 0.0: e:0.0)	No	4.62	1.00	63
cylinder12580[0]_3	Mli_ideal(a: 0.0: e:0.0)	No	4.62	1.00	63
cylinder12580[0]_4	Mli_ideal(a: 0.0: e:0.0)	No	4.62	1.00	63
cylinder12580[0]_5	Mli_ideal(a: 0.0: e:0.0)	No	4.62	1.00	63
cylinder12580[0]_6	Mli_ideal(a: 0.0: e:0.0)	No	4.62	1.00	63
cylinder12580[0]_7	Mli_ideal(a: 0.0: e:0.0)	No	4.62	1.00	63
cylinder12580[0]_8	Mli_ideal(a: 0.0: e:0.0)	No	4.62	1.00	63

44 Pestaña de inspección de superficies

Las superficies están identificadas por una descripción, están conformadas por hasta 4 puntos, los que configuran un área. Poseen una representación de sus propiedades ópticas que modificarán la velocidad en que se calientan teniendo en cuenta si es el comienzo o final de la vida útil del material que la compone. Una de las columnas muestra el coeficiente de iluminación que es un factor utilizado para modelar de manera porcentual (0..1) las secciones eclipsadas de la superficie. Se muestran también los nodos con los que se relaciona cada superficie (Harness) y si se está considerando la normal invertida de la superficie para el cálculo térmico o no.

- Filtrado y orden de Superficies

Se los puede ordenar por Nombre o Área eligiendo el “radio button” correspondiente.

- Modificación de las Superficies del Modelo

Entre las opciones posibles se encuentra, en el sector superior izquierdo, el botón “Add Surface” que permite en una nueva ventana, asignar un nombre a un nuevo componente con determinada temperatura, la nueva superficie se listará en la tabla junto a las demás.

Debajo de la tabla de superficies se encuentran las opciones para realizar modificaciones al harness o la composición del modelo.

- Add as harness

Permite relacionar la superficie seleccionada en la tabla con alguno de los nodos del modelo. Para ello, debe seleccionarse la fila correspondiente, presionar el botón “Add as harness”, que abrirá una ventana de diálogo nueva con un combo que lista los id’s de los nodos disponibles, seleccionar uno y luego presionar “Ok”.

La interfaz gráfica valida, la duplicidad de las relaciones, es decir que no podrán existir dos relaciones de harness entre un mismo disipador y un nodo. Sin embargo la librería térmica no lo valida podrían existir dos relaciones entre el mismo nodo y componente, el usuario de la LibThermalCore deberá evaluar si es correcto o no.

- Delete harness

Permite borrar la relación entre la superficie seleccionada en la tabla con alguno de los nodos que muestra en la columna de Nodes Ids. Para ello, debe seleccionarse la fila correspondiente, presionar el botón “Delete harness”, que abrirá una ventana de diálogo nueva, con un combo que lista los id’s de los nodos con los que se encuentra relacionado, seleccionar uno y luego presionar “Ok”.

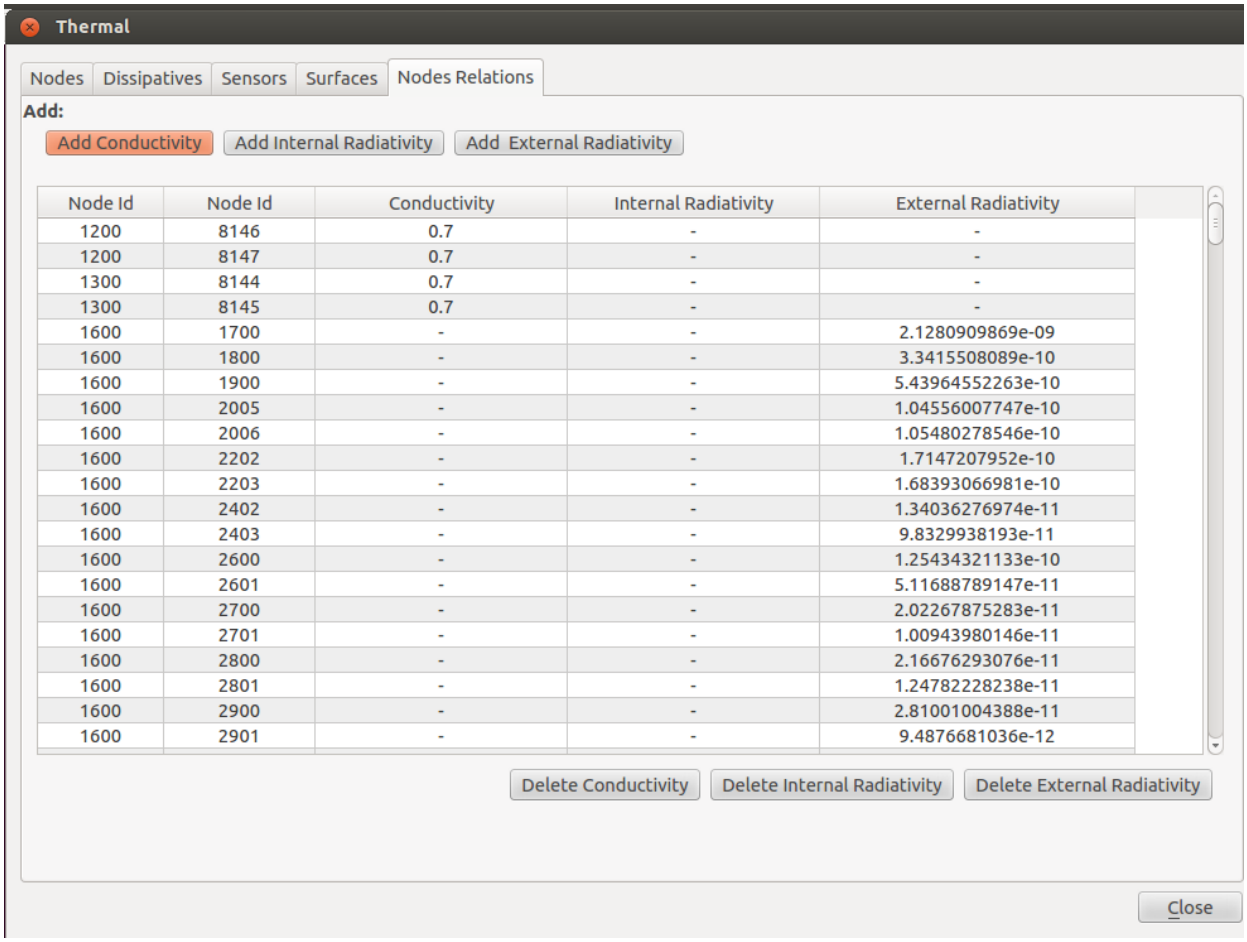
- Delete Surface

Permite borrar la superficie seleccionada en la tabla. Para ello, debe elegirse la fila correspondiente, presionar el botón “Delete Surface”, confirmar la ventana de diálogo y la superficie se removerá de la lista.

El SW valida si está relacionada con algún nodo, en cuyo caso primero el usuario deberá borrar la relación y después el componente

Pestaña Relaciones entre Nodos

En la pestaña “Nodes Relations” se muestran las relaciones entre nodos. Estas pueden ser de tipo “conductivity”, “internal radiativity” y “external radiativity”.



Cada fila representa el par de nodos relacionados y, en cada columna, se indican los valores que ponderan esas relaciones. Si existiera más de una relación de determinado tipo; se muestran a modo de lista. En caso de no existir una relación de determinado tipo, esa columna muestra “-”.

- Modificación de Relaciones Entre Nodos del Modelo

En el sector superior, sobre la tabla aparecen tres botones. Cada uno permite crear una nueva relación entre dos nodos según el tipo indicado en el botón. De esta forma, una nueva relación de conductividad, por ejemplo, se logra presionando el botón “Add Conductivity”, se abrirá una nueva ventana que permite seleccionar dos nodos y asignar el valor específico a esa relación. Mismo procedimiento para las relaciones de radiatividad internas y externas.

El SW valida que los nodos seleccionados sean diferentes entre sí.

Debajo de la tabla de relaciones se dispone de otros tres botones para borrado. Para eliminar por ejemplo una relación de conductividad, se selecciona el par afectado, y se presiona el botón “Delete Conductivity”. De existir más de una relación de conductividad entre estos nodos, se listarán en la nueva ventana y deberá seleccionarse una en particular y luego presionar el botón “Ok”. Mismo procedimiento para las relaciones de radiatividad internas y externas.

El SW valida que el par seleccionado posea una relación de ese tipo para eliminar.

Capítulo 6: Conclusiones y Trabajos Futuros.

i. Conclusiones

El aporte del STS en el procedimiento de trabajo de los equipos involucrados fue muy importante. Esta herramienta fue la primera creada para debug del trabajo del equipo térmico. Este simulador les permite verificar si los datos exportados del ThermalDesktop son los esperados y si el modelado se hizo correctamente. Los casos de error del modelo (en los que las temperaturas tienden a infinito) pueden encontrarse rápidamente y pueden ser corregidos en memoria para chequear que la simulación avanza con éxito. La secuencia de pasos previa, al traspaso de archivos .csv al equipo de desarrollo para la carga de la base de datos, la compilación del simulador de satélite, ejecución y control del mismo, en el que el desarrollador se veía involucrado por completo; fue simplificada de modo que el equipo térmico hoy, carga los .csv en la base de datos del STS por medio de scripts, ejecuta el simulador térmico, depura el modelo, y envía los .csv corregidos al equipo de desarrollo en pos de su integración con otros simuladores de mayor alcance.

El impacto que generó esta simplificación en las relaciones entre equipos se tradujo en:

- la reducción del re trabajo de ambas partes
- mejorías en el modelo reducido generado luego de observar cómo evoluciona en el tiempo
- reducción de las iteraciones causadas por las correcciones del modelo de temperaturas
- disminución del acoplamiento de los datos causada por la separación de las bases de datos del modelo térmico y de los componentes eléctricos

El hecho de que el mecanismo del cálculo térmico se encuentre implementado como librería, resuelve parte de simuladores satelitales futuros, ya que siempre es necesario representar con mayor o menor detalle las variaciones de temperatura en el sistema. En adelante implica una reducción del esfuerzo asociado al desarrollo de simuladores y reutilización de código. Contribuye a la decisión estratégica de generar simuladores con modelos incrementales y modulares, con funcionalidades genéricas desacopladas en forma de librerías compartidas. La codificación en módulos permite crear un producto que separa los aspectos específicos de la plataforma de los aspectos específicos del modelo térmico, obteniendo así portabilidad.

Desde el punto de vista personal, la experiencia en un desarrollo de este tipo afianzó conocimientos en sistemas de tiempo real, creación de librerías compartidas en Linux, desarrollo de simuladores e integración de los mismos con otros sistemas en funcionamiento, desarrollo incremental en particular a través de COMET, planificación, estimación, estado del arte de herramientas de gestión de proyectos, entre otros.

Respecto de las herramientas utilizadas para la gestión del STS se detectó que son muchas; esta situación se torna perjudicial a la hora de querer realizar la trazabilidad en el proyecto. A futuro sería conveniente trabajar con el conector EA para JIRA que permite relacionar los requerimientos con las tareas, para así, poder completar la trazabilidad de Requerimientos-Tareas-Commits relacionándolos de forma sencilla. También es posible lograr trazabilidad al relacionar TestCases-Tickets-Commits. De esta forma se reduciría el mantenimiento y licencias.

En relación con la planificación y la estimación se extrae como conclusión que es conveniente sobredimensionar las fases de mayor incertidumbre, que en este caso eran la arquitectura y la capacitación. Respecto de la Arquitectura la incertidumbre dio como resultado, cambios en el diseño detallado de las distintas librerías. Respecto de la Capacitación, la incertidumbre conllevó a una subestimación de la curva de aprendizaje y a errores, en algunos casos, de diseño y en otros de implementación.
El uso de una metodología de desarrollo, facilitó la implementación del sistema, permitió comprender y tener una visión clara del software a desarrollar. Sin el uso de la misma, la tarea de mantenimiento podría resultar un trabajo arduo y costoso; sin embargo, hoy es un problema simplificado ya que cada clase o modulo posee responsabilidades desacopladas. Se pudo

identificar que COMET se adaptaba correctamente a la problemática a resolver. Utilizar COMET para un desarrollo de tiempo real fue de gran importancia, ya que sirvió de guía, resolviendo algunos problemas comunes en el diseño de sistemas de este tipo.

ii. Trabajos Futuros

El producto de SW resultado de la tesina es la aplicación stand alone STS, con el objetivo de mejorar el flujo de trabajo entre equipos y reducir errores en etapas tardías de la Ingeniería.
Se proponen las siguientes mejoras al simulador térmico:

- Generalizar su implementación para que pueda simular el comportamiento térmico de otro tipo de sistemas embebidos

  La simulación térmica de los modelos de sistemas embebidos no satelitales, ofrecería los mismos beneficios en la integración del trabajo entre equipos, encargados de otros tipos de proyectos. Otorgando así flexibilidad en el uso de la herramienta debido a la simplicidad con la que se cargan los datos.

  Actualmente el STS no está probado con modelos térmicos de sistemas no satelitales, el motivo de ello es que se dispone de una serie de configuraciones dependientes del estado de despliegue del satélite que deben ser cargadas. Otros sistemas pueden carecer de piezas móviles y por lo tanto no presentar estados de despliegue.

  Para esta propuesta de extensión deben estudiarse las diferencias que pueden ocurrir en la configuración de los entornos físicos en los que los sistemas no satelitales se ejecutarán.

  - Control térmico y alarmas

  Como requerimiento posterior del STS, se agrega la capacidad del simulador térmico de tener un algoritmo o SW encargado del control térmico. Así el simulador dará la posibilidad de testear el algoritmo que define los niveles de alarma dentro del sistema.

Otro producto de la tesina es el módulo de cálculo térmico como librería para que pueda ser integrada fácilmente en simuladores de las diferentes etapas del mismo. En particular se cuenta con la posibilidad de:

- Incluir la Libthermalcore en simuladores OPS y AIV

  Se espera que dentro de los planes de desarrollo de Invap, se generen simuladores de Operación y de Verificación del Satélite SAOCOM. Para estos casos, la librería térmica serviría para modelar la fluctuación de la temperatura dentro del sistema. Este trabajo de integración de la LibThermalCore en simuladores abordados en otras fases del satélite, es un trabajo que a futuro puede ofrecer reutilización de código testeado, reducción del esfuerzo de la implementación del OPS y del AIV; y unicidad en el tratamiento de la temperatura entre diferentes softwares.

- Representar el nivel de actividad de disipadores

  Para hacer la simulación más real, se sugiere el manejo del nivel de actividad de los heaters y cargas. El origen de esta necesidad, es que la potencia disipada por una carga mientras está usando el total de su capacidad operativa, es mayor que si se encuentra funcionando en un porcentaje menor o en estado de reposo. Por este motivo se propone tener en cuenta en el cálculo térmico un coeficiente que represente dicha actividad y que modifique el comportamiento de la temperatura del componente. Esta nueva funcionalidad ofrecería un cálculo más realista para el operador interesado en el comportamiento del satélite bajo alguna condición de estress.

- Exportar/Importar un modelo térmico

Esta funcionalidad propuesta como trabajo futuro permitiría al operador generar el modelo reducido a través de la interfaz gráfica del STS y guardarlo para que estén disponibles en otro momento. El formato en el que se exporten debería facilitar la carga de los datos en la BD térmica, lo cual facilitaría la recuperación de la simulación cuando el operador desee.

Apéndices

Apéndice A – Especificación de Requerimientos de SW

i. Introducción

Propósito

El objetivo de este documento es describir el alcance del producto STS - Simulador Térmico Satelital y Librería Térmica, este conjunto de requerimientos deberían ser los conductores del documento de diseño del proyecto

Este documento será actualizado cuando ocurran eventos relevantes que impacten en el alcance descrito.

Alcance

Este documento contiene requerimientos técnicos del SW STS y LibThermalCore.

Este documento es aplicable al diseño de la ingeniería de los mismos.

ii. Descripción general

Funciones y propósito

El STS es una aplicación stand-alone que permite simular el modelo térmico de un satélite.

El propósito del STS es validar el cálculo térmico.

La librería del modelo térmico libthermalcore contiene la ecuación de cálculo térmico y se comunica con la base de datos que provee los nodos a evaluar en la simulación.

La librería puede ser compilada tanto con el STS como con otro simulador.

El STS y la ThermalCore implementan las siguientes funciones:

- Implementación de Simulador y servicios de Scheduler y TimeKeeper.
- Modelo térmico: desarrollo de infraestructura del modelo térmico (kernel de las ecuaciones de térmicas). Incorporación del modelo reducido de plataforma de SAOCOM en la primera iteración.
- Base de datos de cargas, heaters, termistores y termostatos.
- Base de datos del modelo térmico.
- API dedicada a la configuración del modelo térmico.
- API dedicada al control y simulación del modelo térmico.
- HMI para inspección del modelo.
- Desarrollos generales de inspección del modelo para testing.

iii. Consideraciones ambientales

Recursos

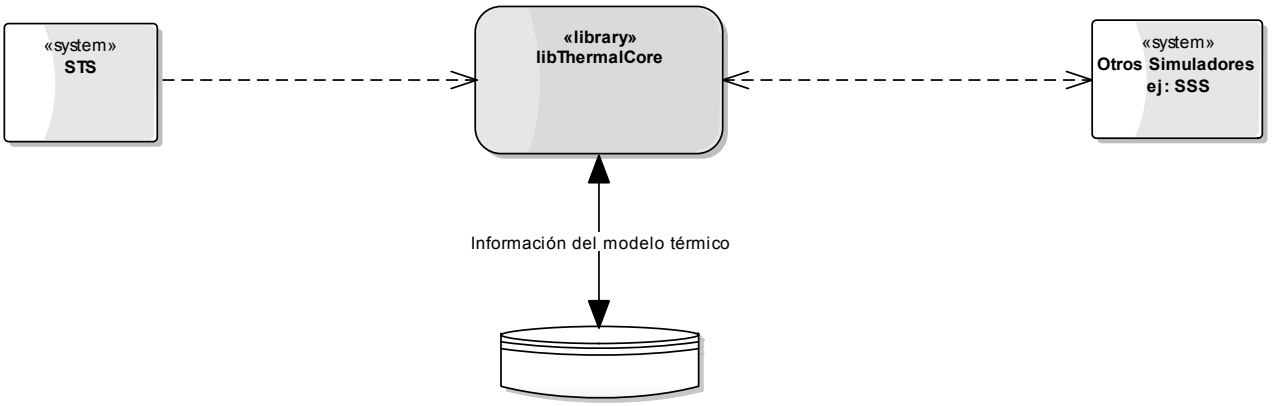
El STS hará uso de los siguientes recursos de código propietario: libsmp2 y libthermalcore.

Los datos se almacenarán en dos bases de datos SQLite, donde una mantendrá el modelo térmico reducido y la otra los datos de los componentes físicos que lo conforman.

El STS correrá en un servidor Ubuntu 12.04 64 bits.

Relación con otros sistemas

Diagrama de Contexto de la LibThermalCore



45 Diagrama de contexto Libthermalcore

- Otros Simuladores ej.: SSS

Simuladores de mayor alcance que hacen uso de la librería.

- STS

STS Simulador Térmico Satelital permite conocer la evolución del modelo térmico controlando una simulación y las variables que modifican el entorno físico en el que se encontraría el satélite (posición del sol, orientación del satélite respecto del mismo, nivel de absorción de la radiación del sol, etc.)

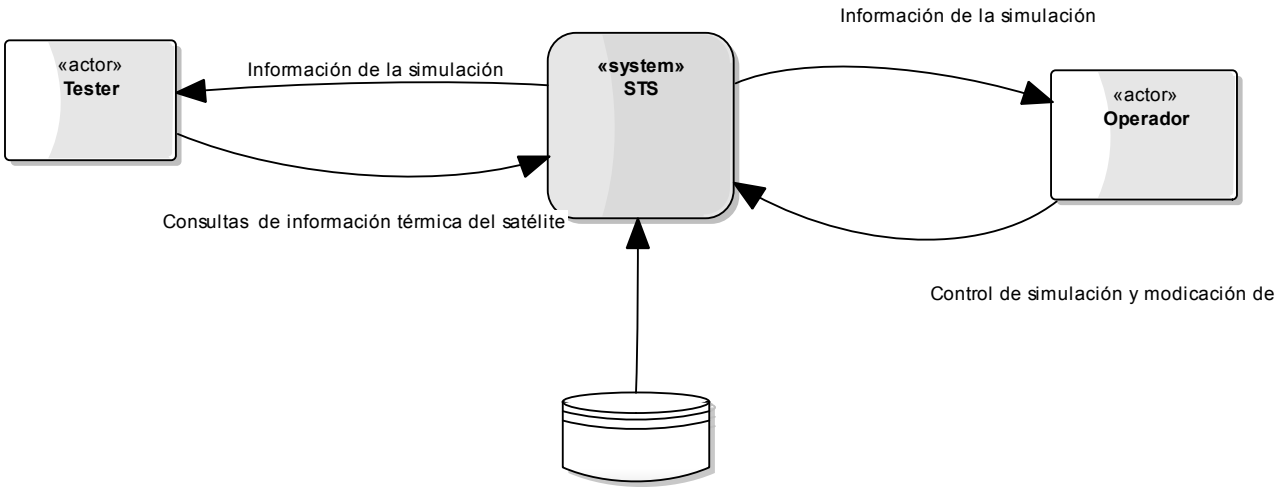
- ThermalDB

Contiene el modelo térmico, conformado por nodos, relaciones de harness, relaciones entre nodos, superficies y puntos, etc.

- LibThermalCore

LibThermalCore se encarga de calcular la temperatura de las partes del modelo térmico de un paso a otro de una simulación.

Diagrama de Contexto del STS



46 Diagrama de contexto STS

- STS DB

Contiene datos de cargas, heaters, termistores y termostatos.

- Operador

Persona que utiliza el simulador como herramienta para desarrollar su trabajo.

- Tester

Persona que realiza pruebas funcionales.

- STS

STS Simulador Térmico Satelital.

iv. Descripción del modelo lógico

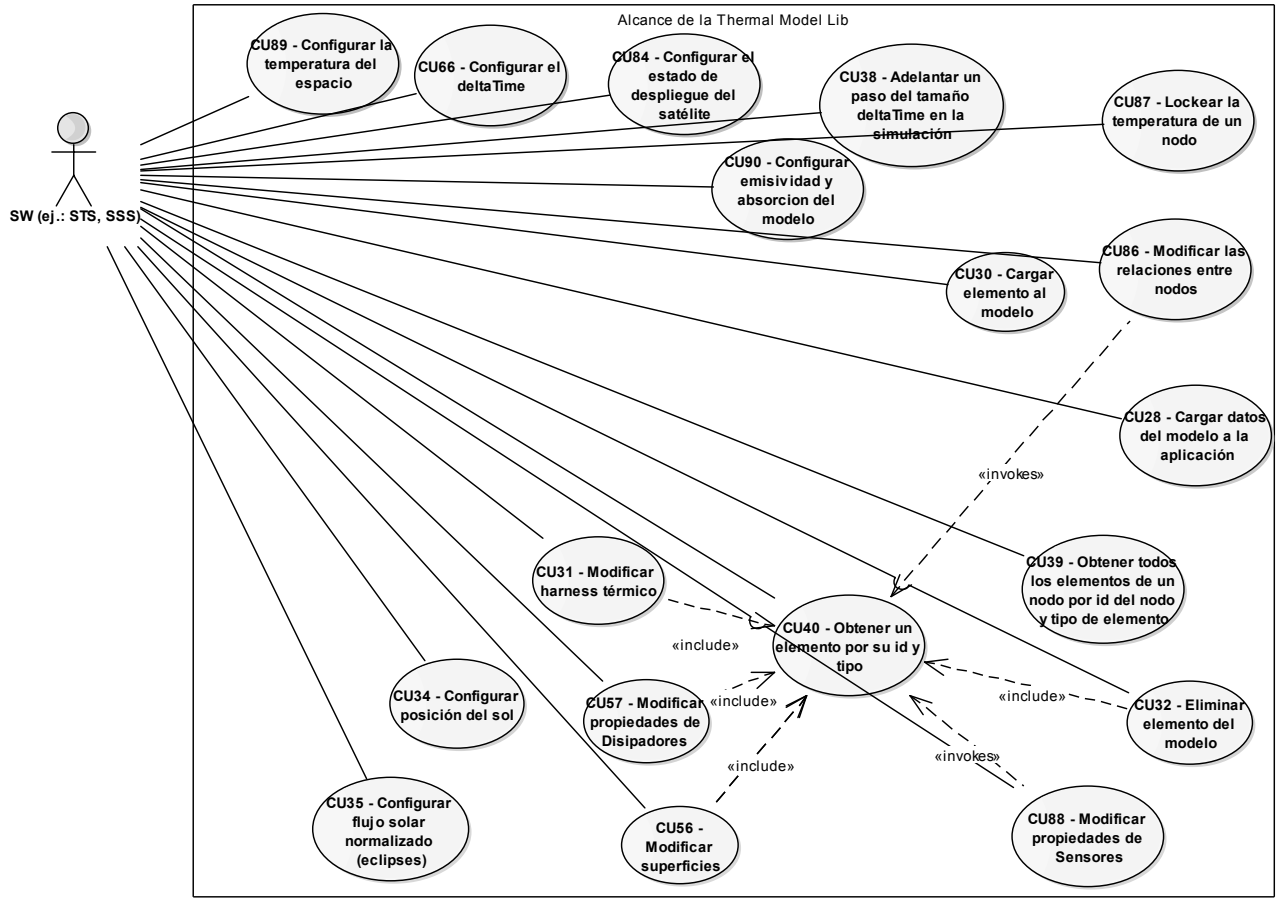
LibThermalCore

Los diagramas de Caso de Uso muestran la interacción entre los actores y el sistema.

Especifican los requerimientos externos.

Cada Caso de Uso define el comportamiento de algún aspecto de la librería sin revelar su estructura interna.

Diagrama CU - Descripción Global

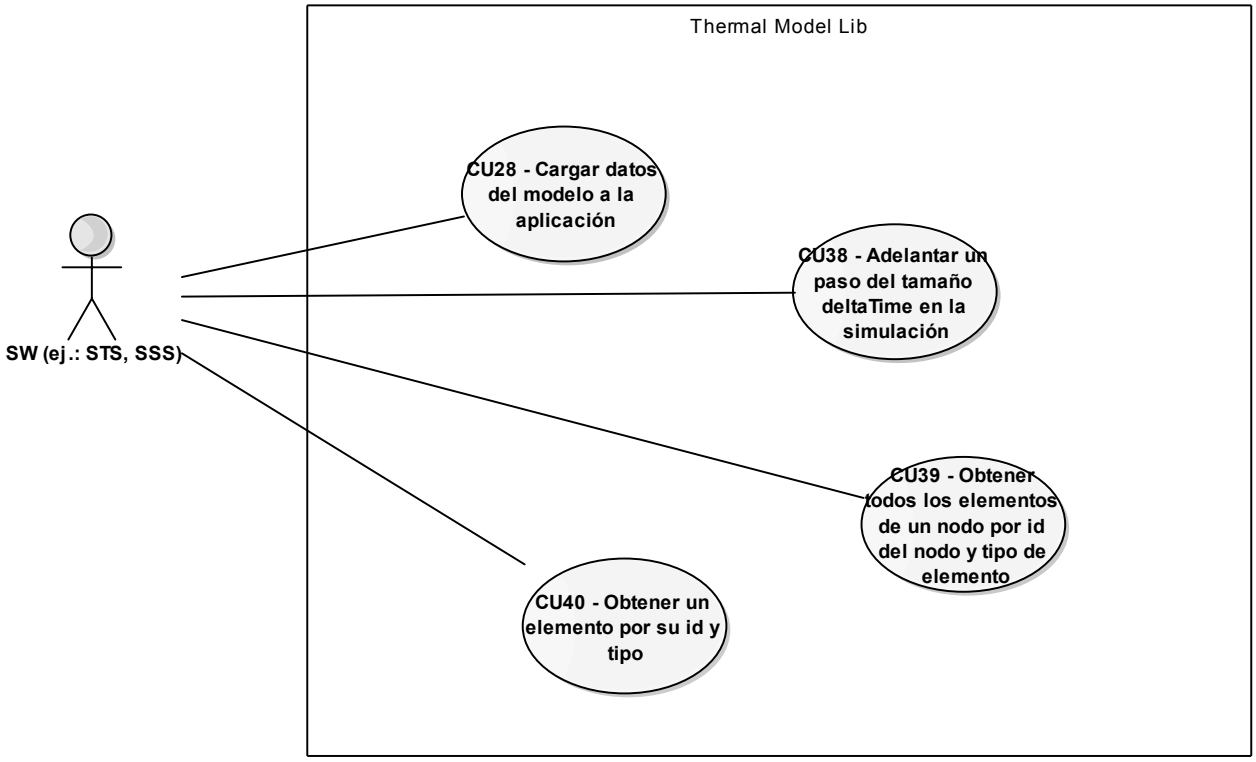


47 LibThermalCore - Visión de Conjunto

- SW (ej.: STS, SSS)

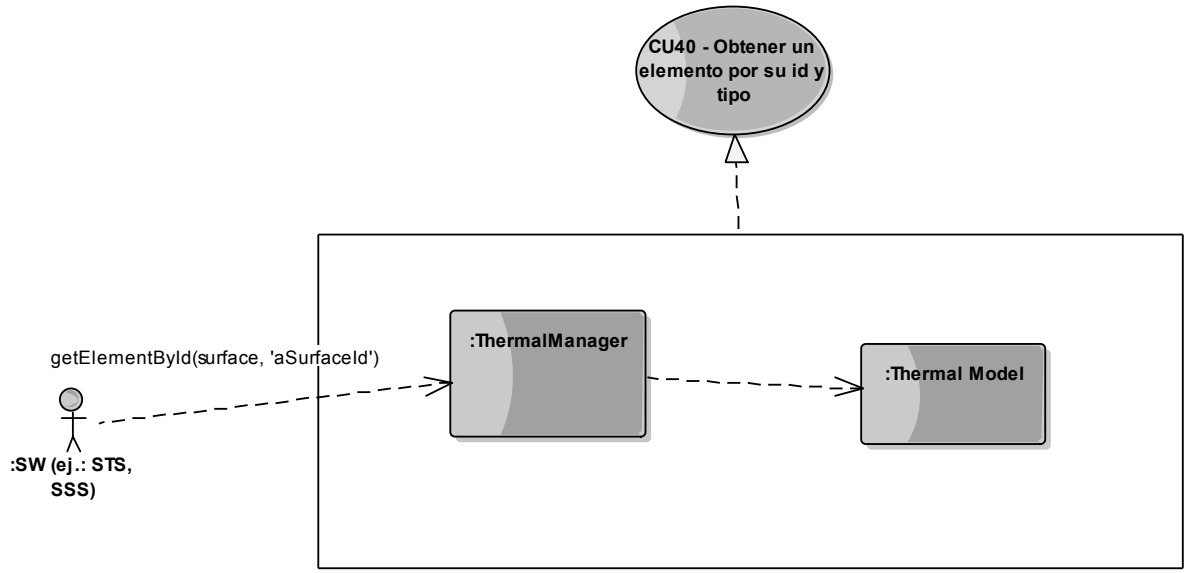
Estos actores, ya sea el STS, u otros simuladores desarrollados que incluyan el cálculo térmico del satélite deben utilizar esta librería e indicarle cuáles son los componentes correspondientes al modelo térmico precargado.

Diagrama CU - Funciones generales



48 Diagrama CU - Funciones generales

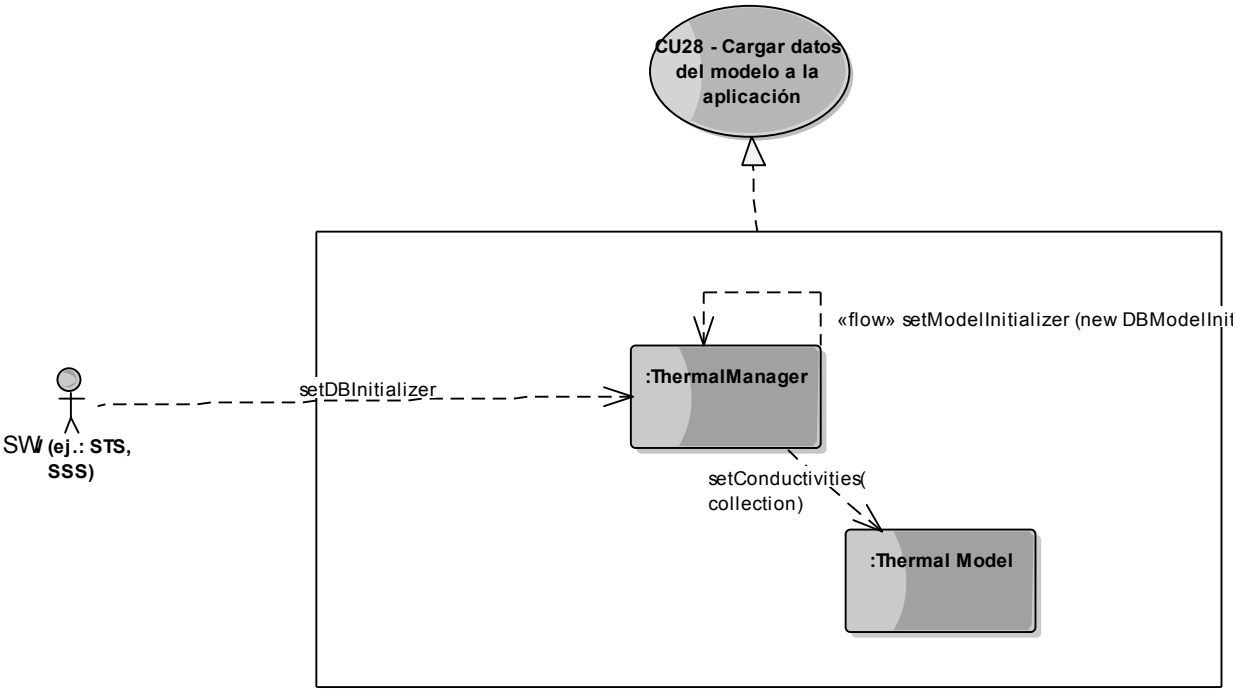
Nombre	**CU40 - Obtener un elemento por su id y tipo**
Observaciones	Consulta por un elemento del modelo
Precondiciones	El parámetro type puede tomar los siguientes valores: node,surfaceHarness, dissipativeHarness, sensorHarness, surface, dissipativeRelatedComponents, sensorRelatedComponents. El parámetro elemID podrá repetirse entre tipos. El modelo debe estar cargado en memoria.
Basic Path	1 - User - Llama al procedimiento "getElementByID(type, elemID)" 2 - System - Retorna un elemento de tipo "type" con identificador ID 2a - Alternate - No existe tal elemento. Retorna en End 1 - System - Retorna código de error ELEM_NOT_FOUND



49 Diagrama de comunicación CU40 Obtener un elemento por su id y tipo

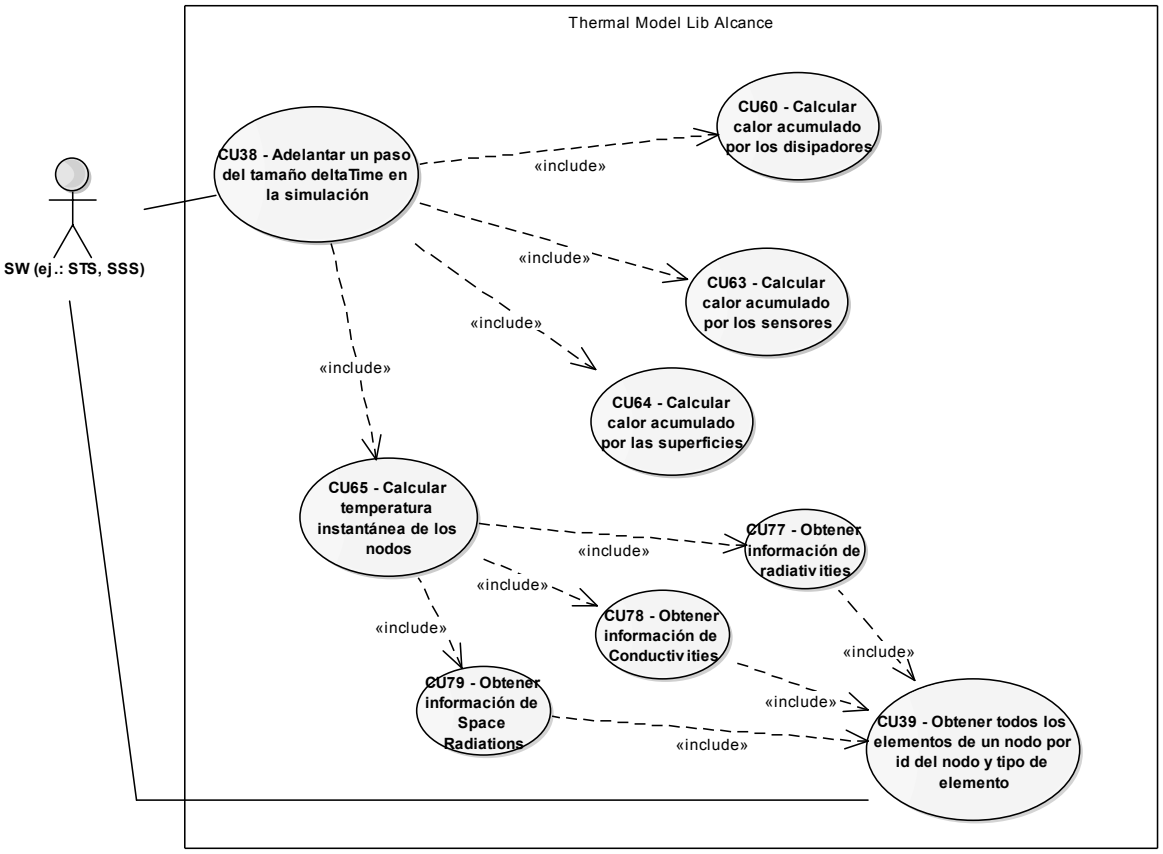
Nombre	**CU39 - Obtener todos los elementos de un nodo por id del nodo y tipo de elemento**
Observaciones	Consulta por una lista de objetos del modelo
Precondiciones	El parámetro type puede tomar los siguientes valores: radíativitiesInternas, radíativitiesExternas, conductivities, harness térmico de termostatos y termistores (sensores), de cargas o heaters (disipadores) o de superficies. La función puede hacer una indirección y por lo tanto el parámetro type también puede ser: surfaceRelatedComponents, dissipativesRelatedComponents o sensorsRelatedComponents. El parámetro nodeId debe existir en el modelo.
Basic Path	1 - User - Llama al procedimiento "getRelatedElementsByNodeID(type, nodeID)" 2 - System - Retorna una lista de elementos de tipo "type" relacionados con el nodo "nodeID" 2a - Alternate - No existen elementos relacionados de ese tipo. Retorna en End 1 - System - Retorna una lista vacía

Nombre	**CU28 - Cargar datos del modelo a la aplicación**
Observaciones	Se cargan los objetos en memoria.
Precondiciones	Existe la Thermal DB con un modelo térmico almacenado.
Basic Path	1 - User - Llama al procedimiento "initialize(dbFilePath)" 2 - System - Se cargan los datos de la base de datos ubicada en path, como objetos en el entorno de ejecución 2a - Alternate - No encuentra el archivo de la base de datos. Retorna en End 1 - System - Muestra código de error DB_ERROR
Post Condiciones	Los datos que se cargan desde la BD a la aplicación representan un modelo térmico. La BD de encuentra en la dirección path, para conectarse se requiere de un usuario user, y contraseña pass.



50 Diagrama de comunicación CU28 Cargar datos del modelo a la aplicación

Nombre	**CU38 - Adelantar un paso del tamaño deltaTime en la simulación**
Observaciones	Este CU permite el avance en la simulación
Precondiciones	La variación de datos ocurre por cambios en la posición del sol, cambios introducidos por modificación de harness, evolución del modelo o elementos de harness y por modificación del estado de despliegue elegido. Al menos existe un modelo con un estado de despliegue cargado en memoria. Se encuentra configurado un deltaTime que le da la duración en tiempo al paso de simulación.
Basic Path	1 - User - Llama al procedimiento "nextStep()" 2 - System - Ejecuta CU63 - Calcular la temperatura acumulada por el Thermal Harness 3 - User - Ejecuta CU60 - Calcular potencia disipada acumulado por el Thermal Harness 4 - User - Ejecuta CU64 - Calcular temperatura acumulada por las superficies 5 - System - Ejecuta CU65 - Calcular temperatura instantánea de los nodos 6 - System - Obtiene la temperatura instantánea actualizada de los nodos teniendo en cuenta el harness térmico
Post Condiciones	La temperatura instantánea calculada se mantiene en memoria



51 Detalle del CU38 Adelantar un paso del tamaño deltaTime en la simulación

Nombre	**CU77 - Obtener información de radiativities**
Observaciones	Este CU permite el avance en la simulación.
Precondiciones	El modelo está cargado en memoria.
Basic Path	1 - User - Con type= radíativitiesInternas y nodeId = nodoi se ejecuta el CU39 - Obtener todos los elementos de un nodo por id del nodo y tipo de elemento 2 - System - Se obtiene una lista con los nodos relacionados a nodoi y el valor en j/k que cuantifica esa relación de radiatividad interna 3 - User - Con type= radíativitiesExternas y nodeId = nodoi se ejecuta el CU39 - Obtener todos los elementos de un nodo por id del nodo y tipo de elemento y se obtienen las radiativities externas existentes en el estado de despliegue actual 4 - System - Se obtiene una lista con los nodos relacionados a nodoi y el valor en j/k que cuantifica esa relación de radiatividad externa
Post Condiciones	La lista de radiativities externas será diferente para el mismo nodo, dependiendo del estado de despliegue actual configurado en la librería.

Nombre	**CU78 - Obtener información de Conductivities**
Precondiciones	El modelo está cargado en memoria.
Basic Path	1 - User - Con type= conductivities y nodeId = nodoi se ejecuta el CU39 - Obtener todos los elementos de un nodo por id del nodo y tipo de elemento 2 - System - Se obtiene una lista con los nodos relacionados a nodoi y el valor en j/k que cuantifica esa relación

Nombre	**CU79 - Obtener información de Space Radíations**
Precondiciones	El modelo está cargado en memoria.
Basic Path	1 - User - Con type= spaceRadíations y nodeId = nodoi se ejecuta el CU39 - Obtener todos los elementos de un nodo por id del nodo y tipo de elemento 2 - System - Se obtiene una lista con el objeto spaceRadíation relacionado al nodoi y el valor en j/k que cuantifica las radiación
Post Condiciones	El resultado variará según el estado de despliegue actual configurado en la librería.

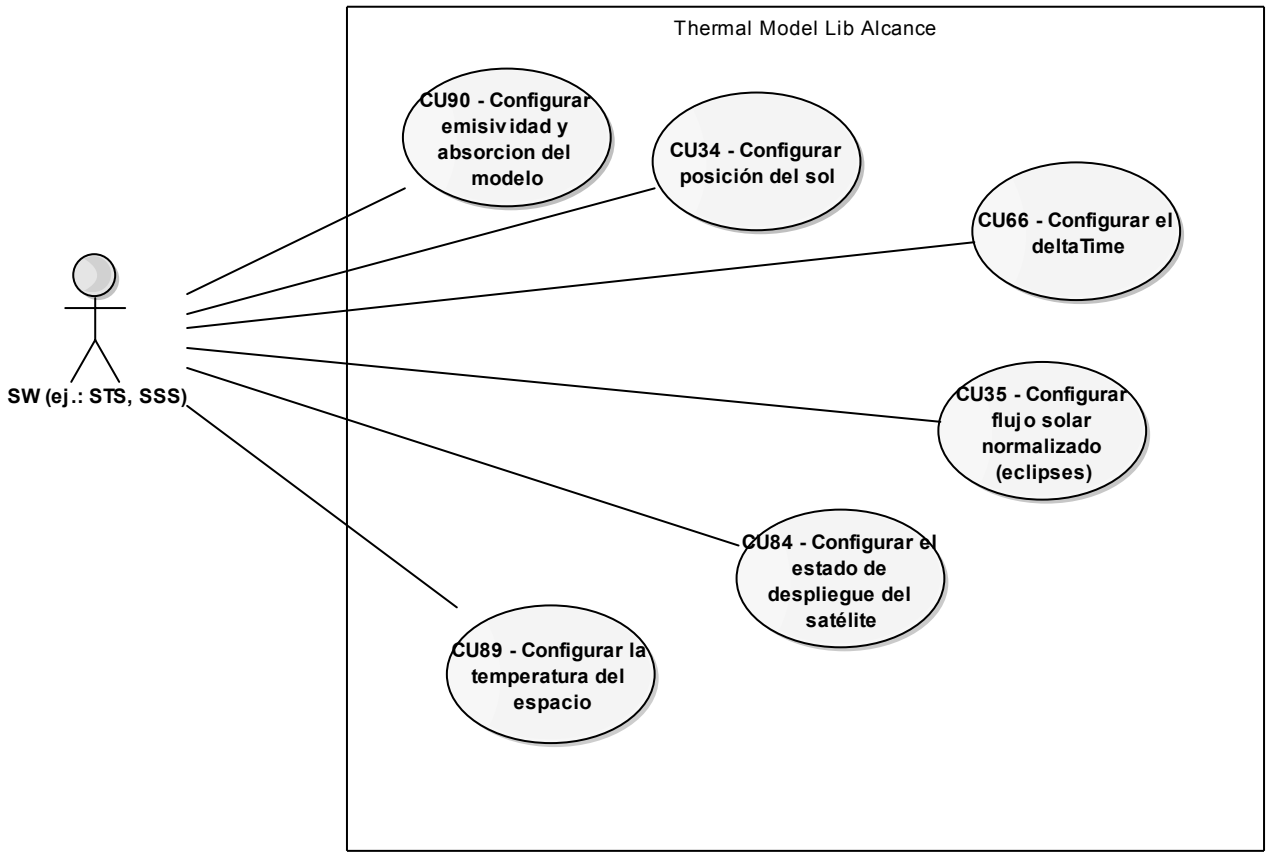
Nombre	**CU60 - Calcular temperatura acumulada por los disipadores**
Observaciones	Existe una tarea que calcula la potencia que se disipa.
Precondiciones	El modelo debe estar cargado en memoria.
Basic Path	1 - User - Llama al procedimiento calculate() del entryPoint DissipativesSolver 2 - System - Para cada nodo, acumula la potencia que disipan elementos del tipo heater o cargas que se encuentren encendidos, para ello debe convertir a temperatura la disipación definida en watts 3 - User - Para cada nodo, actualiza la estructura compartida con el valor calculado
Post Condiciones	La estructura compartida se actualiza con la potencia disipada para cada nodo.

Nombre	**CU63 - Calcular temperatura acumulada por los sensores**
Observaciones	Existe una tarea que se encarga de acumular la temperatura
Precondiciones	Los objetos de harness térmico pueden ser: harness de heaters, harness de cargas, harness de termistores, harness de termostatos. Los elementos de harness asociados a cada objeto de harness pueden estar en on/off y son: heaters, cargas, termistores y termostatos. El modelo se encuentra cargado en memoria.
Basic Path	1 - User - Llama al procedimiento calculate() del entryPoint ThermalHarnessSolver 2 - User - Suma para cada nodo la temperatura que cada elemento de harness asociado y encendido, le aporta 3 - System - Actualiza para cada nodo la estructura compartida con este valor calculado
Post Condiciones	La temperatura calculada se almacenó en la estructura compartida.

Nombre	**CU64 - Calcular temperatura acumulada por las superficies**
Observaciones	Existe una tarea que se encarga de acumular la temperatura de las superficies definidas para el estado de despliegue actual.
Precondiciones	Se encuentra el modelo cargado en memoria.
Basic Path	1 - User - Se llama al procedimiento calculate() del entry point FacetsSolver 2 - System - Calcula para el harness de superficies del estado de despliegue actual, la temperatura acumulada de los nodos, según el flujo solar normalizado, el vector sombra, y el vector sol configurados
Post Condiciones	Se actualiza la estructura compartida con la temperatura instantánea de las superficies. El resultado varía según el estado de despliegue actual.

Nombre	**CU65 - Calcular temperatura instantánea de los nodos**
Observaciones	Existe una tarea que se encarga de resolver la ecuación diferencial
Precondiciones	El modelo se encuentra cargado en memoria
Basic Path	1 - User - Llama al procedimiento calculate() del entryPoint ThermalStep 2 - System - Para un nodoi : Se ejecuta el CU78 - Obtener información de Conductivities 3 - User - Para un nodoi: Se ejecuta el CU77 - Obtener información de radiativities 4 - User - Para un nodoi: Se ejecuta el CU79 - Obtener información de Space Radíations 5 - System - Para un nodoi: se obtiene la temperatura acumulada por el harness térmico, desde la estructura compartida 6 - User - Para un nodoi: se obtiene la temperatura acumulada en las superficies, desde la estructura compartida 7 - System - Para un nodoi: se obtiene la temperatura disipada por el harness térmico, desde la estructura compartida 8 - System - Para un nodoi: se resuelve la ecuación diferencial con los datos de entrada (el deltaTime configurado, la capacitancia del nodo, y la información acumulada en la estructura compartida de QHeater, Qdissipative y QFacet, las radiativities, space radiations, y conductancias). De esta manera calcula la temperatura instantánea del nodoi. 9 - User - Actualiza la estructura compartida con la temperatura calculada para retroalimentar las siguientes iteraciones 10 - System - Repite desde paso 5 tantas veces como nodos tenga el modelo
Post Condiciones	Se actualiza el valor de temperatura instantánea de nodos en la estructura de datos compartida.

Diagrama CU - Configuraciones del entorno



52 Diagrama CU - Configuraciones del entorno

Nombre	**CU84 - Configurar el estado de despliegue del satélite**
Observaciones	State puede tomar los valores GEO (en órbita); GTO (en despliegue); STOWED (en lanzamiento). Por defecto el estado de despliegue es GEO.
Basic Path	1 - User - Llama al procedimiento setDeploymentState(state) 2 - System - Se almacena el nuevo estado de despliegue para tenerlo en cuenta en la resolución de la ecuación 2a - Alternate - El estado de despliegue no existe. Retorna en End 1 - System - Retorna código de error

Nombre	**CU89 - Configurar la temperatura del espacio**
Basic Path	1 - User - Llama al procedimiento setSpaceTemperature(value) 2 - System - El nuevo valor de temperature es value

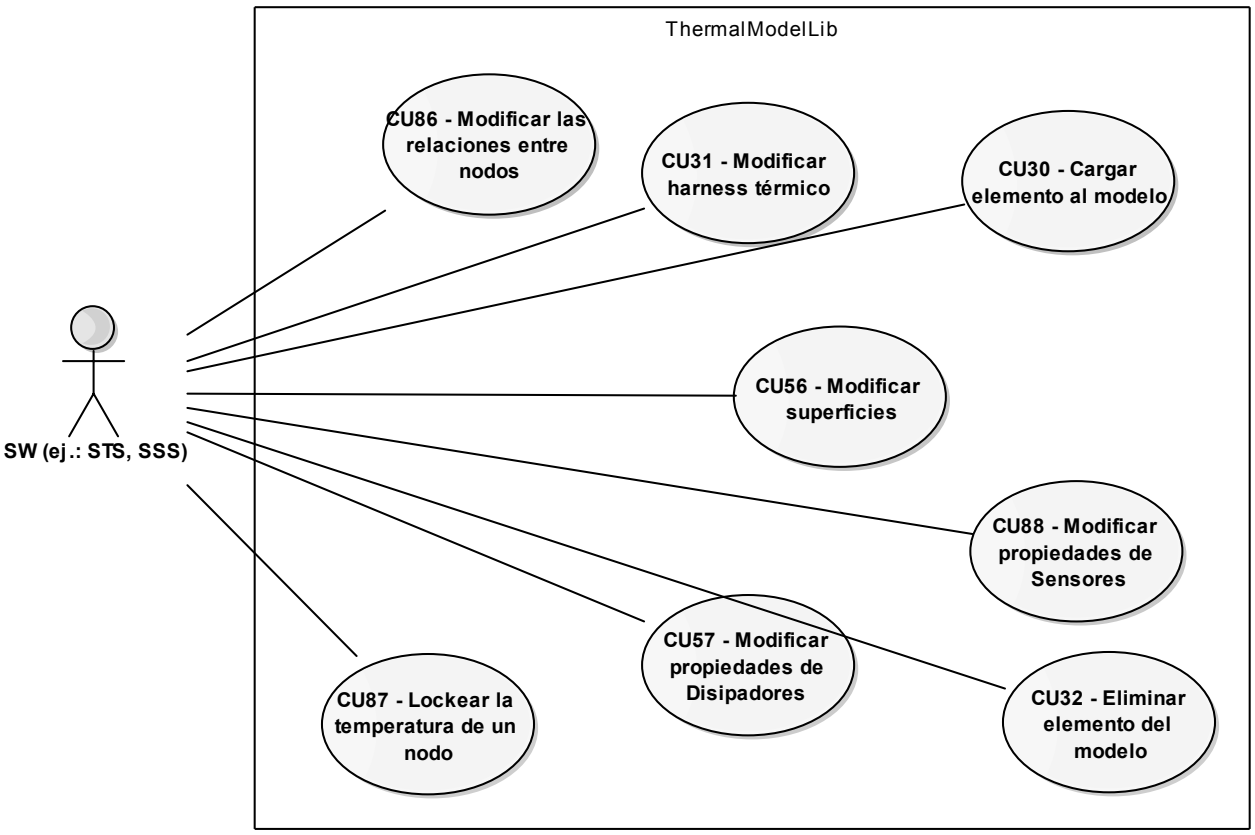
Nombre	**CU90 - Configurar emisividad y absorción del modelo**
Basic Path	1 - User - Llama al procedimiento setEmissivityFactor(valueE) o al setAbsorptivityFacetor (valueA) 2 - System - Los factores toman el valor valueE o valueA según correspondan

Nombre	**CU66 - Configurar el deltaTime**
Observaciones	Asignar un tiempo de avance para los cálculos
Precondiciones	El deltaTime es el tamaño en segundos que dura cada paso de la simulación. Por defecto este valor es 0,2seg. El rango de valores para el parámetro value puede variar de [0,05..0,5].
Basic Path	1 - User - Llama al procedimiento setDeltaTime(value) 1a - Alternate - Value tiene un valor fuera de rango. Retorna en End 1 - System - Retorna código de error
Post Condiciones	Almacena value como deltaTime para ser usado en futuras simulaciones

Nombre	**CU34 - Configurar posición del sol**
Observaciones	El sol modificará la temperatura al que los nodos están sometidos
Precondiciones	La configuración del sol está definida como vector en alguna de las diferentes posiciones de la esfera que contiene al modelo Por defecto no se tiene en cuenta la presencia del sol Value está definido por el vector (x, y, z) puede ser provisto por el DES o bien sea fijo o seteado manualmente en el STS
Basic Path	1 - User - Llama al procedimiento "setSunInBody(value)" 2 - System - Actualiza sunInBody en el Thermal Solver

Nombre	**CU35 - Configurar flujo solar normalizado (eclipses)**
Observaciones	Permite evaluar las caras del satélite con eclipses.
Precondiciones	Se define la normal del flujo solar para poder indicar como afecta la posición del sol en las caras del satélite (generando o no zonas de eclipse). El valor del flujo solar normalizado puede variar de 0..1 siendo 1 la totalidad de incidencia solar. Por defecto este valor es 1. El valor del flujo solar normalizado puede modificarse durante la simulación.
Basic Path	1 - User - Llama al procedimiento "setNormalizedSolarFlux(value)" 1a - Alternate - value está fuera de rango. Retorna en End 1 - System - Retorna código de error
Post Condiciones	Se almacena value y se utiliza para la simulación actual.

Diagrama CU - Configuraciones del modelo



53 Diagrama CU - Configuraciones del modelo

Nombre	**CU86 - Modificar las relaciones entre nodos**
Precondiciones	NodeRelation ya existe en el modelo
Basic Path	1 - User - Ejecuta el CU40 - Obtener un elemento por su id y tipo 1a - Alternate - Llama el procedimiento setInternalRadíativities. Retorna en End 1 - System - Se suma una relación. Cada nodo de la relación genera una referencia al nuevo objeto nodeRelation 1b - Alternate - Llama al procedimiento setConductivitiesK. Retorna en End 1 - System - Se suma una relación. Cada nodo de la relación genera una referencia al nuevo objeto nodeRelation 1c - Alternate - Llama al procedimiento setExternalRadíativitiesPerState. Retorna en End 1 - System - Se suma una relación. Cada nodo de la relación genera una referencia al nuevo objeto nodeRelation 2 - System - Obtiene una referencia a la relación entre nodos 3 - User - Modifica el valor de la relación 3a - Alternate - Llama al procedimiento que elimina la relación. Retorna en End 1 - System - Se eliminan de las referencias a la relación en los nodos involucrados en la misma
Post Condiciones	Se actualiza el valor en el modelo

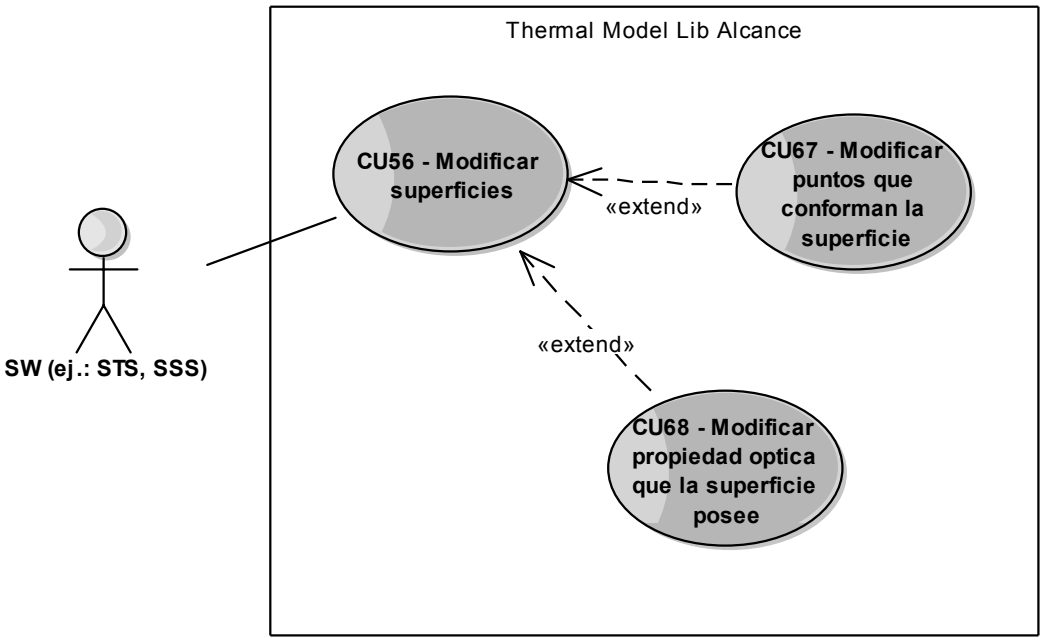
Nombre	**CU87 - Lockear la temperatura de un nodo**
Precondiciones	Se define la normal del flujo solar para poder indicar como afecta la posición del sol en las caras del satélite (generando o no zonas de eclipse). El valor del flujo solar normalizado puede variar de 0..1 siendo 1 la totalidad de incidencia solar. Por defecto este valor es 1. El valor del flujo solar normalizado puede modificarse durante la simulación.
Basic Path	1 - User - Llama al procedimiento lockNodeTemperature con nodeId y temperatureValue 2 - System - Se setea la temperatura de nodeID en temperatureValue y se mantiene el valor durante los siguientes pasos de simulación

Nombre	**CU88 - Modificar propiedades de Sensores**
Observaciones	Modificar la temperatura de elementos del modelo que se encargan de censar el entorno.
Precondiciones	Los sensores térmicos son los termistores, termostatos y termocuplas. La propiedad de un elemento sensor térmico es su temperatura.
Basic Path	1 - User - Llama al procedimiento del CU40 - Obtener un elemento por su id y tipo 2 - User - Modifica la temperatura del sensor obtenido 2a - Alternate - Llama al procedimiento que elimina el sensor. Retorna en End 1 - System - Chequea que el sensor no esté relacionado a un nodo 2 - User - Si está relacionado retorna código de error ya que primero debe borrarse la relación. HAS_RELATIONS 3 - System - De lo contrario se elimina del modelo y retorna código de éxito SUCCESS 2b - Alternate - Llama al procedimiento setSensorComponents. Retorna en End 1 - System - Se agrega la colección de componentes al modelo
Post Condiciones	Se actualiza la temperatura del sensor en el modelo. Las modificaciones de elementos de harness se plasman en memoria, hasta que se almacene la simulación

Nombre	**CU30 - Cargar elemento al modelo**
Precondiciones	Los elementos pueden ser o nodos, o superficies, o harness térmico de heaters, termostatos, termistores o cargas. Elem no existe en el modelo.
Basic Path	1 - User - Llama al procedimiento "addXXX(elem)" pudiendo ser XXX : puntos, superficies, nodos, relaciones entre nodos, disipadores, sensores. 2 - System - Se agrega elem al modelo

Nombre	**CU32 - Eliminar elemento del modelo**
Precondiciones	Los elementos pueden ser o nodos, o superficies, o harness térmico de heaters, termostatos, termistores o cargas. Elem existe en el modelo.
Basic Path	1 - User - Llama al procedimiento "deleteXXX(elem)" pudiendo ser: superficies, disipadores, sensores, relaciones de harness, relaciones entre nodos. 2 - System - Se elimina elem del modelo

Nombre	**CU56 - Modificar superficies**
Observaciones	Se pueden modificar propiedades de superficies.
Precondiciones	Las propiedades de las superficies son los puntos que la conforman y sus 2 propiedades ópticas. El modelo está cargado en memoria.
Basic Path	1 - User - Llama al procedimiento CU40 - Obtener un elemento por su id y tipo 2 - System - Obtiene la propiedad a modificar y la modifica (Ver CU67 - Modificar puntos que conforman la superficie y CU68 - Modificar propiedades ópticas top/bottom que la superficie posee) 3 - System - Se actualizan las propiedades de elem en el modelo 3a - Alternate - El elemento o su propiedad es Read Only. Retorna en End 1 - System - Retorna código de error
Post Condiciones	Las modificaciones de las superficies se plasman en memoria, hasta que se desee almacenar la simulación



54 Diagrama de comunicación CU56 Modificar superficies

Nombre	**CU67 - Modificar puntos que conforman la superficie**
Observaciones	Se modifica la composición de la superficie.
Precondiciones	PointId existe en el modelo.
Basic Path	1 - System - Llama al procedimiento deletePointId(pointId, numberOfPoint) 1a - Alternate - Llama al procedimiento addPointId(pointId, numberOfPoint). Retorna en End 1 - System - Se agrega pointId a la lista y se incrementa N 2 - User - Se elimina el punto de la lista y disminuye N 2a - Alternate - PointId no existe en el modelo. Retorna en End 1 - System - Retorna código de error 2b - Alternate - La superficie no posee puntos asociados. Retorna en End 1 - System - Retorna código de error
Post Condiciones	La superficie con los valores modificados se mantiene en memoria

Nombre	**CU68 - Modificar propiedad óptica que la superficie posee**
Observaciones	Se puede modificar alguna propiedad óptica.
Precondiciones	La superficie con los valores modificados se mantiene en memoria. PropertyId existe en el modelo.
Basic Path	1 - User - Llama al procedimiento CU40 - Obtener un elemento por su id y tipo para obtener una propiedad óptica 2 - User - Modifican propiedades de la propiedad óptica 3 - System - Se actualizan las propiedades en el modelo
Post Condiciones	Si el punto modificado forma parte de una superficie, se verán afectados los resultados del cálculo térmico

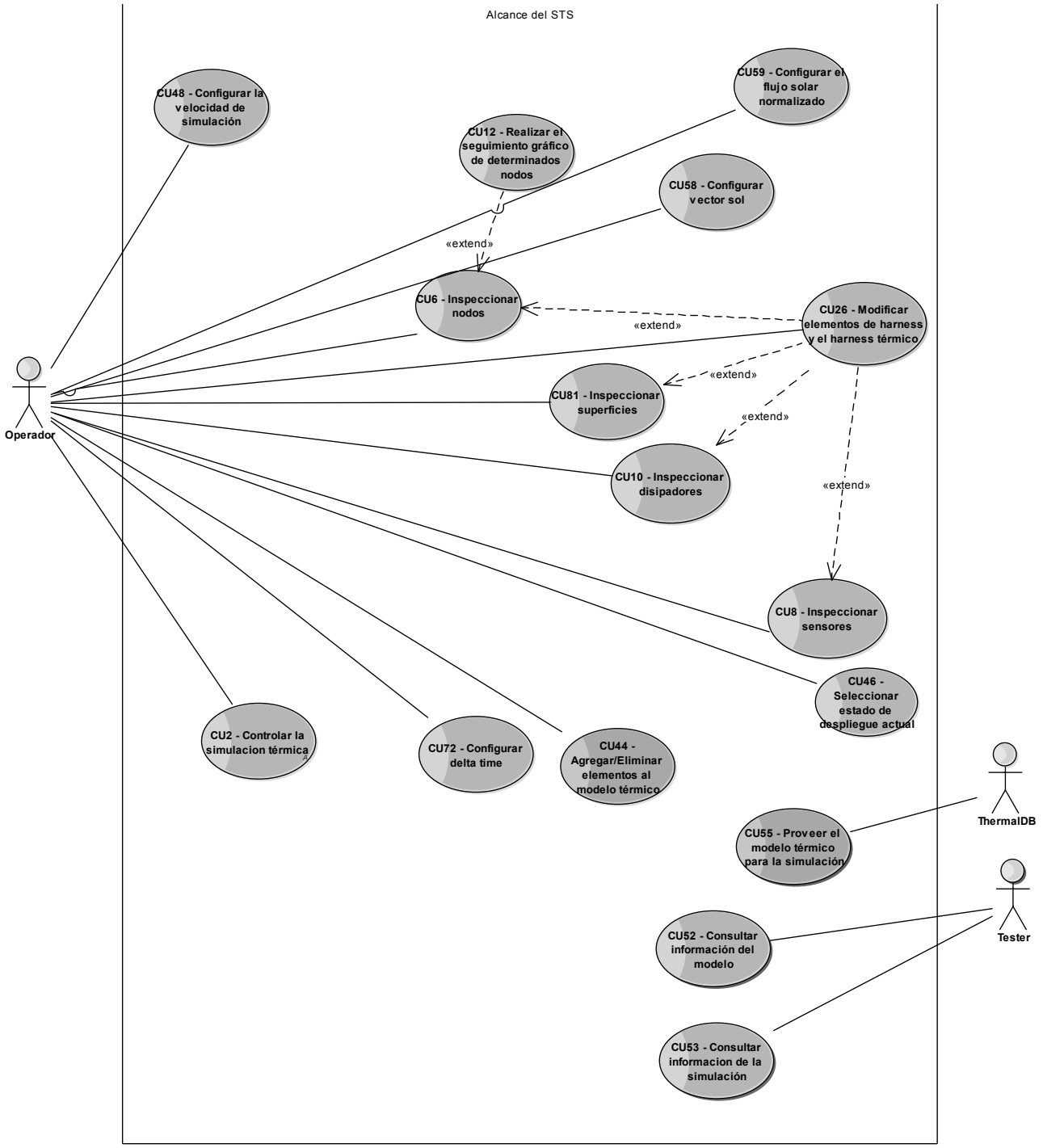
Nombre	**CU57 - Modificar propiedades de Disipadores**
Observaciones	Modificar la temperatura y disipación de los elementos de harness que disipan potencia.
Precondiciones	Los elementos de harness térmico disipadores son los heaters y cargas. La propiedad de un elemento disipador térmico es su temperatura y disipación.
Basic Path	1 - User - Llama al procedimiento del CU40 - Obtener un elemento por su id y tipo

	1a - Alternate - Llama al procedimiento setSensorComponents. Retorna en End 1 - System - Se agrega la colección de componentes al modelo 2 - User - Modifica la temperatura del disipador obtenido 2a - Alternate - Llama al procedimiento que elimina el disipador. Retorna en End 1 - System - Chequea que el disipador no esté relacionado a un nodo 2 - User - Si está relacionado retorna código de error ya que primero debe borrarse la relación. HAS_RELATIONS 3 - System - De lo contrario se elimina del modelo y retorna código de éxito SUCCESS 2b - Alternate - Modifica la cantidad de watts disipados. Retorna en End 1 - System - Se actualiza la disipación del disipador en el modelo
Post Condiciones	Se actualiza la temperatura del disipador en el modelo. Las modificaciones de elementos de harness se plasman en memoria, hasta que se almacene la simulación.

Nombre	**CU31 - Modificar harness térmico**
Observaciones	Se actualizan valores de objetos R/W.
Precondiciones	Existen harness térmicos de heaters, de termostatos, de termistores y de cargas. Los nodos del modelo no se modifican. Las propiedades de harness son su temperatura y potencia que disipan. Elem ya existe en el modelo. El elemento y sus propiedades son R/W.
Basic Path	1 - User - Ejecuta el CU40 - Obtener un elemento por su id y tipo 1a - Alternate - Llama al procedimiento que elimina la relación. Retorna en End 1 - System - Se eliminan las referencias y relaciones creadas por el modelo: se elimina la referencia al objeto desde el modelo, la referencia al componente desde el nodo afectado por el mismo y de la tabla de actualización de temperatura instantánea usada para realizar los cálculos 1b - Alternate - Llama al procedimiento setHarness. Retorna en End 1 - System - Se actualiza el modelo con el nuevo harness 2 - User - Obtiene de esta manera una referencia al objeto y así puede modificar la propiedad deseada. Ej.: nodeId type; componentMnemonic 3 - System - Se actualizan las propiedades de elem en el modelo 3a - Alternate - El elemento o su propiedad es Read Only. Retorna en End 1 - System - Retorna código de error READ_ONLY
Post Condiciones	La modificación de elem queda plasmada en memoria, no se persiste en la base de datos hasta que desee almacenarse la simulación.

STS

Diagrama CU - Descripción Global



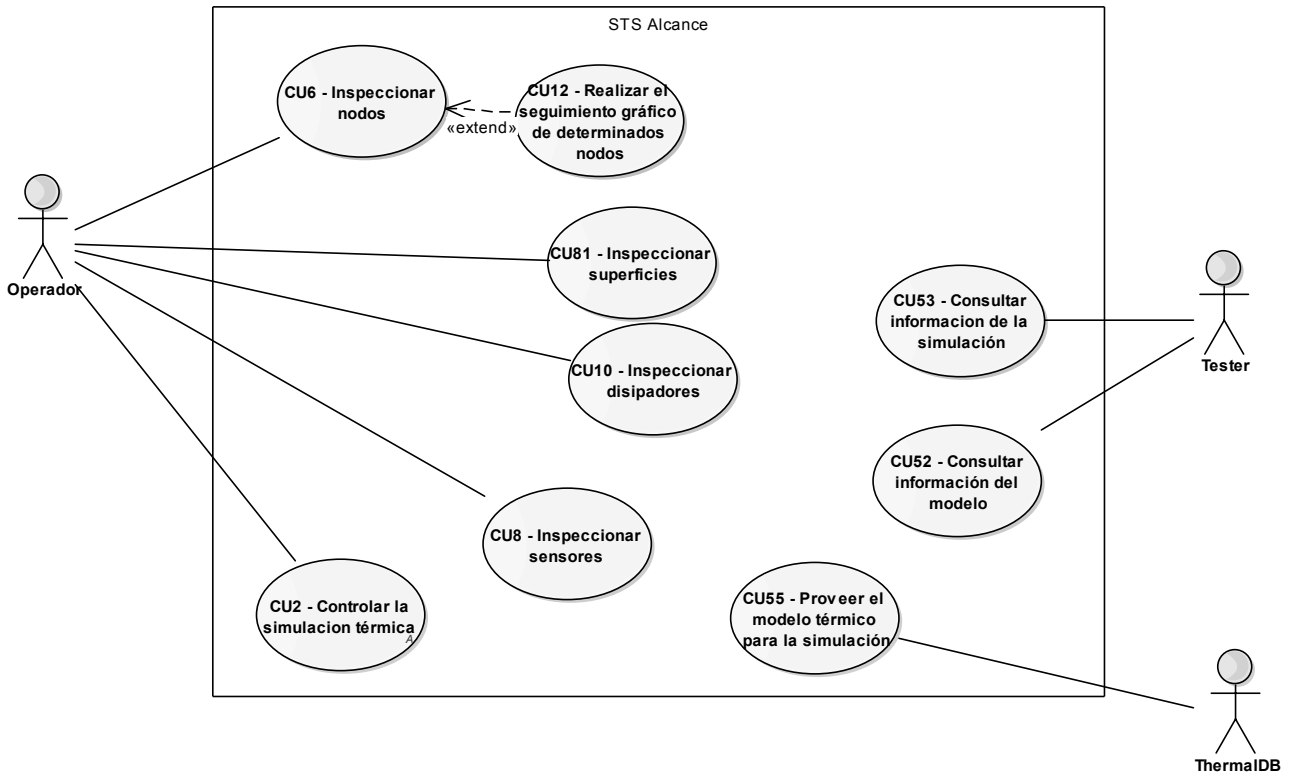
55 Simulador Térmico Satelital - Visión de Conjunto

- Tester

  Es quien interactúa con el sistema a través de scripts python o c++.
- Operador

  Es quien interactúa con el sistema a través de su interfaz gráfica.

Diagrama CU - Funciones generales



56 Diagrama CU - Funciones generales

Nombre	**CU55 - Proveer el modelo térmico para la simulación**
Observaciones	La definición de los componentes de un modelo se almacena en la Thermal DB.
Precondiciones	Los datos se cargan con una herramienta externa que ejecuta un script para cargar la información de los archivos extraídos del TD2DSS.
Basic Path	1 - System - Conectar a la BD y cargar objetos a memoria

Nombre	**CU81 - Inspeccionar superficies**
Precondiciones	Las superficies están formadas por N puntos y tienen propiedades ópticas. Y se relacionan con cierto conjunto de nodos, lo cual se llama harness de superficies. El modelo está cargado en memoria.
Basic Path	1 - User - Accede al menú principal 2 - System - Elige la opción inspeccionar superficies 3 - User - Lista para cada superficie: nombre, los nodos relacionados, puntos que la conforman, coeficiente de iluminación y sus propiedades ópticas. Muestra la opción de ordenar por nombre o por área

Nombre	**CU12 - Realizar el seguimiento gráfico de determinados nodos**
Observaciones	Se muestra una gráfica de los nodos seleccionados en la tabla de inspección.
Precondiciones	El modelo está cargado en memoria
Basic Path	1 - User - Ejecuta el CU5 2 - System - Muestra la opción de abrir la herramienta Graph 3 - User - Selecciona la abrir Graph 4 - System - Abre Graph en la parte inferior de la ventana principal 5 - User - Selecciona de la tabla el o los nodos y presiona el botón “Add” 5a - Alternate - Selecciona Clear. Retorna en End 1 - System - Limpia todas las series del gráfico 5b - Alternate - Selecciona Remove Curve. Retorna en End 1 - System - Quita alguno de los nodos graficados 6 - System - Comienza a graficar la evolución de la temperatura instantánea de ese nodo
Post Condiciones	El gráfico muestra una curva de la temperatura que adquiere el nodo

Nombre	**CU6 - Inspeccionar nodos**
Observaciones	Se listan las propiedades de los nodos de la simulación.
Precondiciones	El modelo está cargado en memoria.
Basic Path	1 - User - Accede al tab Nodos 2 - System - Lista para cada nodo: temperatura instantánea, subgrupo del satélite al que pertenece, componentes asociados (Harness Térmico: sensores , superficies o disipadores), cuánta temperatura aporta el subconjunto de disipadores de cada nodo (Qdissipation) y si se encuentra lockeada su temperatura 3 - User - Obtiene información de los nodos

Nombre	**CU10 - Inspeccionar disipadores**
Observaciones	Se listan las propiedades de los disipadores de la simulación. Los disipadores incluyen: las loads y los heaters.
Precondiciones	Las loads y heaters son híbridos: generan y disipan potencia cuando están encendidos El modelo está cargado en memoria.

Basic Path	1 - User - Accede al menú principal 2 - System - Elegir en el submenú inspeccionar disipadores 3 - System - Lista para cada disipador: nombre, los nodos relacionados, la disipación y la temperatura instantánea. Muestra la opción de ordenar por nombre, disipación, o temperatura eligiendo el “radio button” correspondiente.

Nombre	**CU8 - Inspeccionar sensores**
Observaciones	Se listan las propiedades de los termostatos y termistores de la simulación.
Precondiciones	El modelo se encuentra cargado en memoria
Basic Path	1 - User - Accede al tab Sensors 2 - System - Lista para cada sensor: nombre, nodos relacionados, temperatura que censa, Se muestran 2 radio button para ordenar por nombre o temperatura

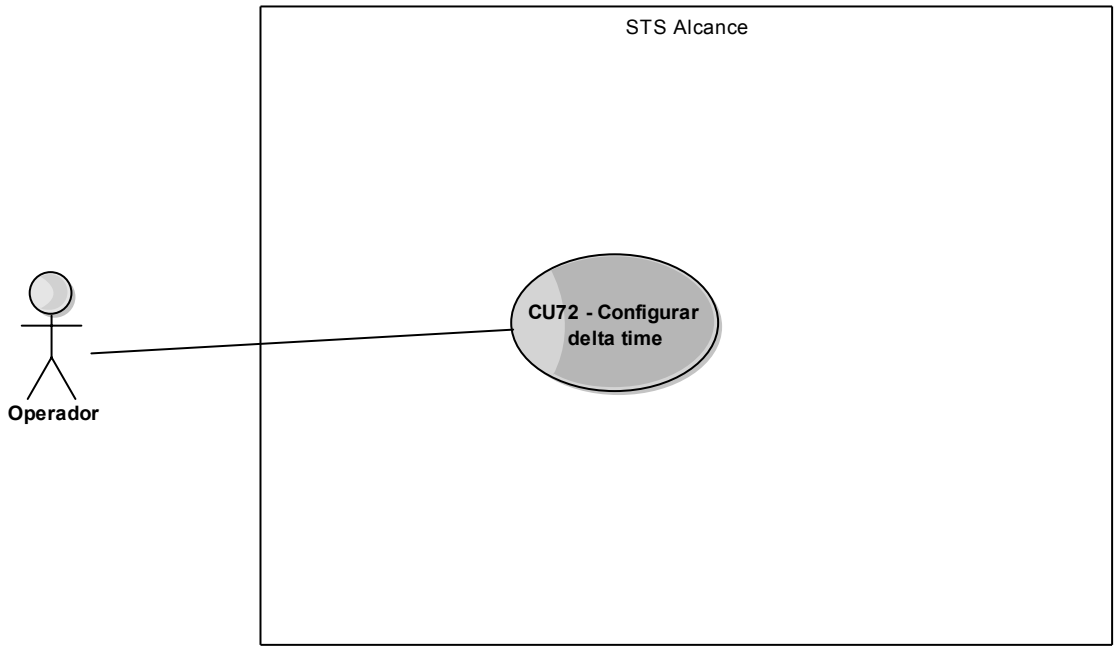
Nombre	**CU2 - Controlar la simulación térmica**
Observaciones	El operador puede comenzar, pausar, parar y ver cantidad de pasos ejecutados por la simulación térmica.
Precondiciones	La simulación se realizará para el estado de despliegue actualmente seleccionado. La simulación se ejecutará a la velocidad indicada por el factor velocidad, que indica si será más rápida o lenta respecto del tiempo del servidor respectivamente. La simulación tendrá en cuenta el flujo solar normalizado, el sol configurado y, en consecuencia las sombras. La simulación tendrá en cuenta la temperatura acumulada por el harness térmico en cada paso El tiempo de simulación es la cantidad de pasos del tamaño deltaTime que se ejecutaron desde el start hasta el stop de la simulación. Existe un timer que determina cuándo se cumple el deltaTime y se modifica según el factor velocidad Precondición: Debe haber un modelo precargado Precondición: No hay simulaciones ejecutándose el estado del simulador es Standby
Basic Path	1 - User - Selecciona Start, comienza a correr la simulación. 2 - System - En la sección de la interfaz gráfica "Simulation Status" se muestra el estado Executing 3 - System - Se ejecuta un paso (llama al procedimiento nextStep() de la librería) con una periodicidad calculada a partir del delta time y el factor

	velocidad que se estén configurados 4 - System - Se incrementa el tiempo de la simulación, y se repite desde el paso 3 hasta que ingrese otro comando 5 - User - Selecciona Pause 6 - System - La simulación se detiene y se muestra el estado de simulación Standby, los datos de las variables y el tiempo de simulación se mantienen en memoria 7 - User - Selecciona Start, comienza a correr la simulación. 8 - System - Se ejecuta un paso (llama al procedimiento nextStep() de la librería) con una periodicidad calculada a partir del delta time y el factor velocidad que se estén configurados 9 - System - Se incrementa el tiempo de la simulación, y se repite desde el paso 8 hasta que ingrese otro comando 10 - User - Selecciona Stop 11 - System - La simulación se detiene y el tiempo de simulación y las estructuras compartidas se resetean, el simulador pasa al estado StandBy

Nombre	**CU52 - Consultar información del modelo**
Precondiciones	El modelo se encuentra cargado en memoria
Basic Path	1 - User - Consulta información a través de alguna de las funciones de thermalcoreclientapi 2 - System - El STS retorna los datos que fueron solicitados

Nombre	**CU53 - Consultar información de la simulación**
Precondiciones	Debe estar corriendo una simulación
Basic Path	1 - User - Consulta información a través de alguna de las funciones de libstsclientapi 2 - System - El STS retorna los datos que fueron solicitados

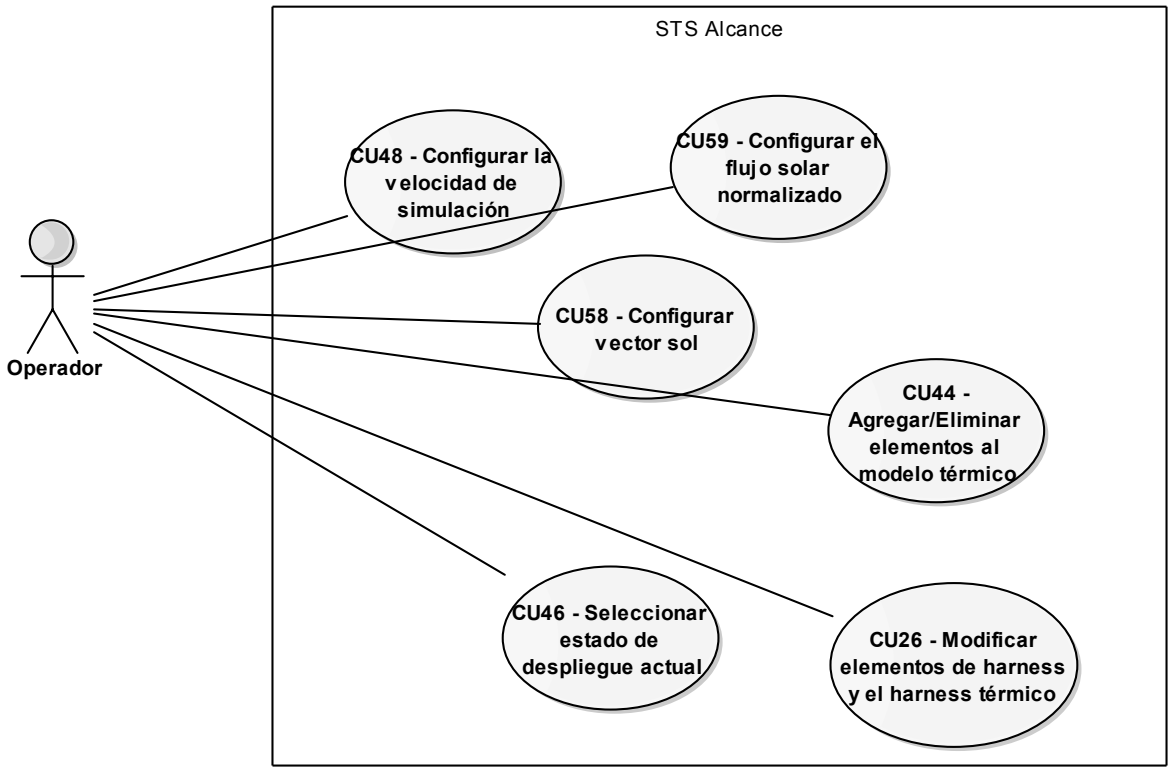
Diagrama CU - Configuraciones posibles sólo en estado StandBy



57 Diagrama CU - Configuraciones posibles sólo en estado StandBy

Nombre	**CU72 - Configurar delta time**
Observaciones	Se modifica el tamaño de los pasos de la simulación
Precondiciones	El simulador debe estar en estado StandBy para modificar este valor.
Basic Path	1 - User - En el menú principal elige la opción DeltaTime 2 - System - Muestra un cuadro de diálogo para completar el valor del delta time 3 - User - Completa con el nuevo valor y presiona Accept 3a - Alternate - Presiona Cancel. Retorna en End 1 - System - No modifica el valor del delta time anterior
Post Condiciones	Se almacena el nuevo valor para futuras simulaciones

Diagrama CU - Configuraciones en simulación



58 Diagrama CU - Configuraciones en simulación

Nombre	**CU58 - Configurar vector sol**
Precondiciones	El estado de despliegue puede modificarse durante la ejecución de la simulación. Los vectores del sunInBody se encuentran cargados en la Thermal DB.
Basic Path	1 - User - En el menú principal se accede a la opción setSunInBody 2 - System - Se muestra una lista/gráfico de los vectores solares 3 - User - Selecciona un vector y presiona OK 3a - Alternate - Selecciona Cancel. Retorna en End 1 - System - No se modifica el valor sunInBody
Post Condiciones	Se modifican los valores de sombra según la nueva configuración del vector solar Se actualiza el valor sunInBody en la simulación automáticamente

Nombre	**CU59 - Configurar el flujo solar normalizado**
Precondiciones	El flujo del sol normalizado se usa en el cálculo de las zonas de eclipse
Basic Path	1 - User - Ejecuta el CU58 - Configurar vector sol 2 - System - En la misma ventana se muestra la opción de editar el valor del flujo solar normalizado pudiendo varia de 1 a 0. 3 - User - Modifica el valor del flujo 4 - System - Se actualiza con el nuevo valor automáticamente

Nombre	**CU26 - Modificar elementos de harness y el harness térmico**
Observaciones	Si la variable permite sobre escritura, se le puede asignar algún valor.
Precondiciones	Los elementos del modelo que se pueden modificar son: - las superficies, el harness térmico de heaters, termistores, termostatos y cargas, propiedades de heaters, termistores, termostatos y cargas La variable posee permisos de escritura
Basic Path	1 - User - Selecciona de la lista la variable, completa con el New Value en la propiedad y presiona Override 1a - Alternate - Ingresa un valor de tipo incorrecto o fuera de rango y presiona override. Retorna en 1 1 - System - Muestra un cuadro de diálogo indicando el error y no se sobrescribe el valor de la variable
Post Condiciones	Se muestra el nuevo valor en la variable ubicada en la lista

Nombre	**CU46 - Seleccionar estado de despliegue actual**
Observaciones	El estado de despliegue implica variantes en la incidencia del sol sobre las caras del satélite
Precondiciones	Por defecto el estado de despliegue es Geo El modelo tiene al menos un estado
Basic Path	1 - User - Accede al menú principal, opción Select Deployment State 2 - System - Abre una ventana de diálogo donde se lista las distintas opciones de estados de despliegue 3 - User - Selecciona un estado y presiona Accept

	3a - Alternate - Presiona Cancel. Retorna en End 1 - System - No se modifica el estado de despliegue actual y se cierra la ventana de diálogo
Post Condiciones	Se actualizan las listas de elementos del modelo para ese estado de despliegue Se muestran las radiativities externas, space radíations y geo points específicos del estado de despliegue

Nombre	**CU44 - Agregar/Eliminar elementos al modelo térmico**
Basic Path	1 - User - En el menú principal selecciona AddXXX... 2 - System - Abre una ventana de diálogo que permite completar las características propias del elemento según sea una relación entre nodos, una superficie, un disipador o sensor 3 - User - Completa y presiona Ok 3a - Alternate - Presiona Cancel. Retorna en End 1 - System - No se crea el nuevo elemento
Post Condiciones	Se crea el nuevo elemento en el modelo

Nombre	**CU48 - Configurar la velocidad de simulación**
Observaciones	Configurar la velocidad de simulación
Precondiciones	El factor de velocidad de simulación puede modificarse durante la simulación según SMP2. Por defecto el Speed Factor = 0,5x, Simulation Time/step = 0. Este factor puede ser variar entre [4.0x .. 5.0x].
Basic Path	1 - User - Accede a la opción de configuración de tiempo y completa nuevos valores 2 - System - Utiliza estos valores para la simulación actual

v. Requerimientos

Requerimientos funcionales

Requerimientos Funcionales del Modelo Térmico

- **Requerimiento60 - La temperatura del satélite deberá incluir el flujo solar**

Tipo: «Functional» El modelo térmico tendrá en cuenta la posición del sol respecto de las superficies y sus nodos componentes en el cálculo de temperatura.

- **Requerimiento86 - La temperatura del satélite deberá tener en cuenta las conductancias**

Tipo: «Functional» El modelo térmico deberá tener en cuenta las conductancias como relaciones entre nodos para el cálculo de temperatura

- **Requerimiento87 - La temperatura del satélite deberá incluir las capacitancias**

Tipo: «Functional» El modelo térmico tendrá en cuenta las capacitancias como como representación de cada nodo para el cálculo de temperatura

- **Requerimiento15 - La temperatura del satélite deberá incluir zonas de eclipse**

Tipo: «Functional» El modelo térmico tendrá en cuenta los efectos de los eclipses en el cálculo de temperatura

- **Requerimiento58 - La temperatura del satélite tendrá que distinguir caras internas**

Tipo: «Functional» El modelo térmico tendrá en cuenta las relaciones entre nodos en las superficies internas (radíatividades internas) y externas (radíatividades externas) para el cálculo de la temperatura

- **Requerimiento48 - La temperatura del satélite deberá distinguir caras externas y disipación espacial**

Tipo: «Functional» El modelo térmico tendrá en cuenta las interacciones radíantes entre nodos externos del modelo reducido y el nodo de temperatura de espacio, es decir, la radiación con el espacio para el cálculo de la temperatura.

- **Requerimiento51 - La temperatura del satélite deberá incluir el efecto térmico de loads**

Tipo: «Functional» El modelo térmico tendrá en cuenta el efecto del harness térmico de las Loads para el cálculo de la temperatura

- **Requerimiento50 - La temperatura del satélite deberá incluir el efecto térmico de termistores**

Tipo: «Functional» El modelo térmico tendrá en cuenta los efectos del harness térmico de los termistores para el cálculo de la temperatura

- **Requerimiento10 - La temperatura del satélite incluye el efecto térmico de termostatos**

Tipo: «Functional» El modelo térmico deberá tener en cuenta los elementos de control pasivo como termostatos. Los efectos de su harness térmico en los nodos del modelo para el cálculo de la temperatura.

- **Requerimiento17 - La temperatura del satélite deberá incluir el efecto térmico de heaters**

Tipo: «Functional» El modelo térmico tendrá en cuenta el harness térmico de heaters definido en la Thermal DB. El harness térmico de heaters debe relacionar un heater con el o los nodos a los cuales transmite temperatura. El modelo térmico lo tendrá en cuenta para el cálculo de la temperatura.

- **Requerimiento65 - La temperatura del satélite deberá incluir el efecto térmico de propiedades ópticas**

Tipo: «Functional» Se tendrán en cuenta los niveles de degradación de las propiedades ópticas de las superficies para el cálculo de la temperatura

Requerimientos Funcionales de la LibThermalCore

- **Requerimiento38 - Thermal Model Lib API deberá permitir leer la Thermal DB**

Tipo: «Functional» La API permitirá cargar los datos de la BD a la aplicación

- **Requerimiento79 - La Thermal Model Lib deberá permitir la configuración de la posición del sol**

Tipo: «Functional» La Thermal Model Lib usará la ubicación del sol para realizar el cálculo térmico

- **Requerimiento78 - La Thermal Model Lib deberá Incluir zonas de eclipse en el cálculo**

Tipo: «Functional» La Thermal Model Lib usará el flujo solar normalizado para representar zonas de eclipse

- **Requerimiento75 - La Thermal Model Lib tendrá que proveer la posibilidad de editar QHeater y QDissipative**

Tipo: «Functional» La Thermal Model Lib permitirá gestionar la potencia que irradía o disipa el harness térmico asociado a un nodo

- **Requerimiento46 - Thermal Model Lib deberá hacer el cálculo de la temperatura del satélite**

Tipo: «Functional» La Thermal Model Lib calculará la variación térmica de los nodos de un tiempo T1 a un tiempo T2 teniendo en cuenta posición del sol respecto del modelo, eclipse, sombras y el harness térmico asociado al nodo en determinado tiempo

- **Requerimiento53 - Thermal Model Lib deberá ocultar la conexión a la Thermal DB**

Tipo: «Functional» La conexión a la Thermal DB será transparente para el SW que utilice la librería

- **Requerimiento91 - La Thermal Model Lib deberá tener en cuenta el estado de despliegue**

Tipo: «Functional» La Thermal Model Lib realizará la ecuación térmica de un paso a otro teniendo en cuenta el estado de despliegue que se le configure

Requerimientos Funcionales del STS

- **Requerimiento22 - La interfaz gráfica deberá permitir usar filtros o búsquedas**

Tipo: «Functional» La inspección de elementos debe permitir el orden y/o filtrado de los mismos al menos por un criterio

- **Requerimiento29 - STS deberá permitir configurar tiempos**

Tipo: «Functional» El STS permitirá configurar el tiempo delta para definir los pasos de simulación

- **Requerimiento32 - STS deberá permitir el testing de modelos térmicos**

Tipo: «Functional» El STS permitirá al Operador testear el modelo térmico generado a partir del Thermal Desktop

- **Requerimiento35 - STS deberá simular la evolución térmica del satélite en el tiempo**

Tipo: «Functional» El STS de un paso a otro, debe evolucionar la simulación térmica del satélite en base a la configuración cargada: según el modelo y modificaciones del usuario

- **Requerimiento61 - Una simulación deberá poseer tiempos**

Tipo: «Functional» El simulador permitirá gestionar la velocidad de ejecución de la simulación

- **Requerimiento89 - El STS deberá permitir inspeccionar información de despliegue**

Tipo: «Functional» El STS permitirá ver información térmica de un estado de despliegue por vez

- **Requerimiento67 – El STS deberá permitir modificar los datos del modelo térmico**

Tipo: «Functional» El STS permitirá sobrescribir valores del modelo térmico

- **Requerimiento68 - Una simulación deberá ocurrir en un determinado estado de despliegue**

Tipo: «Functional» El STS permitirá simular variaciones de temperatura para el modelo térmico reducido de un satélite en un determinado estado de despliegue

Requerimientos de rendimiento

- **Requerimiento73 – El STS deberá permitir configurar velocidad**

Tipo: «Performance» Se debe poder configurar la velocidad de ejecución de una simulación. Es decir, poder simular el paso de determinado tiempo en la simulación en un período diferente de tiempo real.

Requerimientos de interface

- **Requerimiento2 – Deberá existir la Thermal Model Lib API**

Tipo: «Constraint» La Thermal Model Lib debe proveer una API.

- **Requerimiento40 - Thermal Model Lib API deberá exponer la gestión de nodos**

Tipo: «Functional» La API de la Thermal Model Lib permitirá gestionar nodos (obtener uno, obtener sus vecinos)

- **Requerimiento41 - Thermal Model Lib API deberá exponer la gestión del Sol**

Tipo: «Functional» La API de la Thermal Model Lib permitirá configurar la ubicación del Sol

- **Requerimiento42 - Thermal Model Lib API deberá exponer la gestión de eclipse**

Tipo: «Functional» La API de la Thermal Model Lib permitirá gestionar el flujo solar normalizado (para representar zonas de eclipse)

- **Requerimiento47 - Thermal Model Lib API deberá exponer la gestión de harness**

Tipo: «Functional» La API de la Thermal Model Lib permitirá gestionar la potencia que irradía o disipa el harness térmico asociado a un nodo

- **Requerimiento80 - Thermal Model Lib API deberá permitir modificar elementos de harness**

Tipo: «Functional» La API de la Thermal Model Lib permitirá gestionar propiedades de los elementos de harness (heaters, termistores, termostatos, cargas) y superficies para introducir fallas en la simulación

- **Requerimiento20 - STS deberá exponer funcionalidad de la Thermal Lib API**

Tipo: «Constraint» El STS debe exponer la API de la Thermal Model Lib

- **Requerimiento90 - El STS facilitara los comandos start, pause, stop**

Tipo: «Functional» El STS proveerá mecanismos para control de la simulación

- **Requerimiento37 - STS deberá poseer una GUI**

Tipo: «Functional» El STS tendrá interfaz gráfica

- **Requerimiento88 - El STS deberá proveer una API**

Tipo: «Functional» El STS proveerá una API

Requerimientos de diseño y restricciones de implementación

- **Requerimiento4 - ThermalModelLib deberá usar ThermalDB**

Tipo: «Constraint» La Thermal Model Lib debe usar la Thermal DB

- **Requerimiento11 - ThermalDB deberá usar SQLite**

Tipo: «Constraint» La Thermal DB debe ser implementada por un motor de base de datos SQLite

- **Requerimiento12 - STS y Thermal Lib APIs deberán ser invocadas en lenguaje C++ o Phyton**

Tipo: «Constraint» La API de la Thermal Model Lib y del STS se implementará en C++ con un wrapper python para que pueda ser llamada por las interfaces HMI (ya sean gráficas o scripts)

- **Requerimiento13 - STS deberá ser implementado en C++**

Tipo: «Constraint» El STS será codificado en C++

- **Requerimiento6 - STS deberá usar la ThermalModelLib**

Tipo: «Constraint» El STS usará la Thermal Model Lib compilada como librería.

- **Requerimiento7 - STS deberá usar la librería SMP2**

Tipo: «Constraint» El STS implementará el estándar SMP2. Usando la SMP2Lib de código propietario

- **Requerimiento9 - STS deberá correr en Linux Ubuntu 64bits**

Tipo: «Constraint» El simulador térmico funcionará sobre un servidor Linux Ubuntu 64 bits.

- **Requerimiento64 - El modelo térmico deberá ser reducido y representativo**

Tipo: «Constraint» El modelo térmico reducido describirá el comportamiento del satélite a partir de un circuito térmico representativo. El mismo será obtenido a partir del modelo global del satélite (de mayor detalle).

- **Requerimiento49 - Nodos, relaciones entre si y harness deberán ser parte de modelo térmico**

Tipo: «Constraint» El modelo térmico estará definido a partir de nodos y sus relaciones entre sí (conductancia y radíactividades), con el espacio (radiaciones espaciales) y con el harness térmico

- **Requerimiento52 - Las superficies deberán ser parte del modelo térmico**

Tipo: «Constraint» El modelo térmico asociará los nodos que lo conforman a superficies geométricas con propiedades ópticas

- **Requerimiento55 – Deberá existir estados para un modelo térmico**

Tipo: «Constraint» El modelo térmico posee diferentes estados de despliegue

Requerimientos de validación

Esquema de validación

Para la verificación de requerimientos se utiliza una o más de los siguientes métodos:

- Prueba

El método de verificación por prueba consiste en la ejecución de Test unitarios y E2E (end to end) utilizando las herramientas correspondientes para el SW desarrollado.

- Inspección

El método de inspección consiste en la verificación visual tanto de la interfaz gráfica como de la documentación para indicar el correcto funcionamiento del SW bajo un determinado criterio.

- Análisis

El método de análisis se utiliza para reemplazar o complementar alguno de los dos anteriores. Consisten en aplicar una evaluación técnica, modelos matemáticos, simulaciones u algoritmos.

Requerimientos de validación

Específicos

- **Requerimiento69 – El SW deberá tener validación con Stakeholders**

Tipo: «Testing» Se evaluarán los resultados de la etapa de análisis con los Stakeholders.

- **Requerimiento70 - El SW deberá tener testing unitario STS & Thermal Model Lib**

Tipo: «Testing» Se realizará testing unitario en el código.

- **Requerimiento71 - El SW deberá tener testing funcional del STS**

Tipo: «Testing» Se realizará testing funcional del simulador operándolo con diferentes conjuntos de datos y se evaluará su rendimiento bajo un entorno conocido.

Métodos de Verificación por requerimiento

Requerimiento	***Validación***
Requerimiento2	Análisis
Requerimiento4	Análisis
Requerimiento6	Análisis
Requerimiento7	Análisis
Requerimiento9	Análisis
Requerimiento10	Prueba
Requerimiento11	Análisis
Requerimiento12	Análisis
Requerimiento13	Análisis
Requerimiento15	Prueba
Requerimiento17	Prueba

Requerimiento	***Validación***
Requerimiento50	Prueba
Requerimiento51	Prueba
Requerimiento52	Análisis
Requerimiento53	Prueba
Requerimiento55	Análisis
Requerimiento58	Prueba
Requerimiento60	Prueba
Requerimiento61	Prueba
Requerimiento64	Análisis
Requerimiento65	Prueba
Requerimiento67	Prueba

Requerimiento20	Inspección
Requerimiento22	Inspección
Requerimiento29	Prueba
Requerimiento32	Análisis
Requerimiento35	Prueba
Requerimiento37	Inspección
Requerimiento38	Prueba
Requerimiento40	Prueba
Requerimiento41	Prueba
Requerimiento42	Prueba
Requerimiento46	Prueba
Requerimiento47	Prueba
Requerimiento48	Prueba
Requerimiento49	Análisis

Requerimiento69	Análisis
Requerimiento70	Análisis
Requerimiento71	Análisis
Requerimiento73	Prueba
Requerimiento75	Prueba
Requerimiento78	Prueba
Requerimiento79	Prueba
Requerimiento80	Prueba
Requerimiento86	Prueba
Requerimiento87	Prueba
Requerimiento88	Inspección
Requerimiento89	Inspección
Requerimiento90	Prueba
Requerimiento91	Prueba

Apéndice B – Descripción del Diseño del SW

Introducción

Propósito

El objetivo de este documento es describir el diseño del producto STS - Simulador Térmico Satelital y LibThermalCore a partir del documento de Especificación de Requerimientos.

Este documento será actualizado cuando ocurran eventos relevantes que impacten en el diseño descrito.

Alcance

Este documento describe la arquitectura del Software. Además especifica aspectos de interfaz tenidos en cuenta para alcanzar los requerimientos.

Capítulo 1: LibThermalCore

Esta sección describe la arquitectura general desde un punto de vista de alto nivel.

General

La librería térmica se encarga de modelar la temperatura de un satélite y sus diferentes partes.

Este desarrollo se diseñó usando conocimientos previos obtenidos del simulador SSS para los satélites ARSAT y SAOCOM, con un diseño basado en clases que refactoriza la implementación conocida para lograr una librería genérica que pueda ser reutilizada para cualquier simulador con el fin de representar la temperatura del mismo sin necesidad de re implementar la funcionalidad térmica cada vez.

El modelo térmico reducido de un satélite emplea nodos para identificar diferentes puntos significativos, que se relacionan entre sí y con el espacio trasmitiendo su calor. Estos se encuentran conectados también a componentes sensores, y disipadores de potencia, y a superficies que modifican la temperatura de su entorno según sus propiedades o estado. Los datos de este modelo se cargan desde una base de datos dedicada.

La librería térmica posee una API también implementada en forma de librería (libthermalcoreclientapi) que provee la posibilidad de realizar llamadas a procedimientos a un servidor (libthermalserverapi) y de obtener información sobre el modelo térmico. Se utilizaron para esta funcionalidad las librerías xmlrpc-c y abyss-server.

Esta API puede operarse por medio de scripts, que hacen las veces de clientes y se comunican con la aplicación a través de los métodos que la libthermalcoreclientapi implementa tanto en C++ como en Python para el control y configuración del modelo.

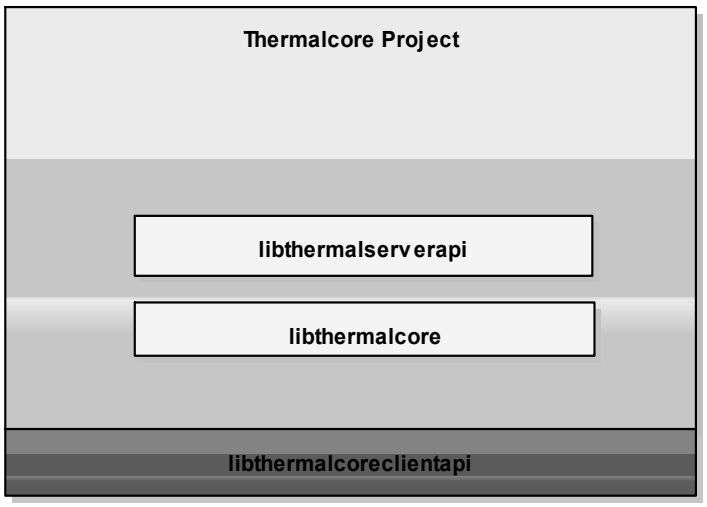
La LibThermalCore fue pensada con el objetivo de reutilizar código que ya fue testeado, reducir el alcance de la compilación en caso de requerir modificaciones en funciones térmicas, optimizar el uso de recursos sacando ventaja del trato que da el sistema operativo a las librerías compartidas (única carga en memoria no importa cuántos programas estén haciendo uso de ella).

En el caso de la LibThermalCore se espera que sea utilizada tanto en el STS como en futuros simuladores que se desarrollen en la empresa.

Estas librerías con funcionalidad dedicada, permiten el desarrollo de una aplicación que separa los aspectos específicos del cálculo y el modelo, del mecanismo de comunicación que permite el acceso a los datos, obteniendo así portabilidad.

Arquitectura Estática del Software

Estructura de la implementación de la LibThermalCore



59 Estructura de la implementación de la LibThermalCore

- libthermalcoreclientapi

Esta librería ofrece la posibilidad de comunicarse con el servidor xmlrpc correspondiente, provee clases cliente que permiten instanciar objetos y serializarlos en las consultas al servidor para modificar el modelo o bien, en sentido contrario es posible representar los objetos del servidor en la memoria del cliente. Puede utilizarse la librería tanto en C++ como en Python, debido a que se utilizó la herramienta SWIG para lograr esta adaptación y facilitar los desarrollos de interfaces HMI por ejemplo.

Una vez implementada esta librería, se realizó testing funcional de la LibThermalCore, por medio de scripts python que en cada método recorrían de manera transversal la arquitectura de librerías desarrolladas: libthermalcoreclient – libthermalserverapi - libthermacore

- Thermalcore Project

Aplicación que permite realizar el cálculo de la temperatura de un satélite en base a un modelo térmico del mismo evolucionando sus partes.

- LibThermalCore

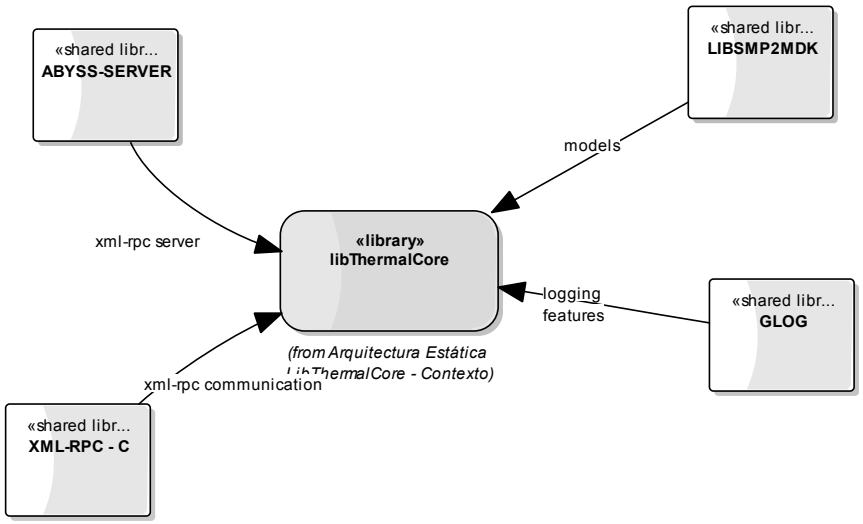
Esta librería expone funcionalidad para inicializar el modelo y avanzar un paso en el cálculo térmico. Inspeccionar diferentes elementos del modelo y modificarlo. Además es posible configurar el tiempo que representa un paso en la simulación, la incidencia del sol, del estado de despliegue y los eclipses sobre la iluminación del satélite.

Durante el desarrollo se realizó testing unitario con CppUnit que permitió, descartar la existencia de errores a medida que se iba a avanzando en las capas superiores de la arquitectura de librerías, es decir, asegurar que el modelo funcionaba correctamente, facilitó detectar errores en el desarrollo de la librería cliente.

- libthermalserverapi

Esta librería ofrece una interfaz y un server xml-rpc, que facilitan la comunicación e interacción de los clientes con la LibThermalCore. De esta manera se encuentra disponible un servidor que responde consultas por protocolo xmlrpc y registra los métodos que llaman a la funcionalidad de la LibThermalCore

Contexto de la implementación de la LibThermalCore



60 Diagrama de contecto de libThermalCore

- ABYSS-SERVER «shared library»

Abyss Web Server es un servidor web compacto disponible para los sistemas operativos Windows, Mac OS X y Linux. Es liviano y es compatible con diferentes protocolos.

- GLOG «shared library»

GLOG Modulo de Google que permite loguear. Está implementado en C++.

- LIBSMP2 «shared library»

Provee la funcionalidad de modelos en el SMP2. Estos modelos pueden contener tareas que se agregan en el scheduler del simulador.

- XML-RPC – C «shared library»

XML-RPC es uno de los enfoques de servicios web más simples que hace que sea fácil que los programas puedan realizar llamadas a procedimientos a través de una red.

Utiliza el protocolo HTTP para pasar información desde un equipo cliente a un equipo servidor.

XML-RPC utiliza un pequeño vocabulario XML para describir la naturaleza de las solicitudes y respuestas, especificando nombre de procedimiento y los parámetros de la petición e identifica ciertos códigos de error.

Funcionalidad

La LibThermalCore cubre las siguientes funciones según se especifican en el SRS.

Un SW (simulador, scripts, etc.) puede:

- Inspeccionar los distintos elementos del modelo del satélite, conociendo su estado y propiedades.
- Configurar la temperatura del entorno en el que se encuentra el satélite.
- Configurar posición respecto al sol y estado de despliegue del satélite.
- Realizar el cálculo térmico y configurar el tamaño del paso de tiempo para el mismo.
- Modificar parte del modelo: sensores, disipadores, superficies, harness y relaciones entre nodos.
- Lockear la temperatura de un nodo en el cálculo.

Estas operaciones se pueden realizar a través de métodos wrappeados a Python o métodos C++ de la libthermalcoreserverapi.

Refactoring de la librería ThermalCore

Tomando como base la funcionalidad térmica existente en los simuladores actuales, se propuso realizar un refactoring del código de manera que sea orientado a objetos, con las responsabilidades más marcadas y menos acoplamiento entre las clases. En la sección "Diseño de Componentes de LibThermalCore“ se detalla el diagrama de clases final.

El diseño de clases estaba acoplado a la estructura de la base de datos, por lo tanto una clase Thermal contenía toda la información para cargar datos a memoria y para calcular la evolución de su temperatura. En consecuencia se separó esta funcionalidad en: un ThermalManager que configura el modelo que se representa con la clase ThermalModel, una clase que hace el cálculo térmico para determinado modelo y un AbstractThermalModelInitializer que tiene la capacidad de brindarle objetos al modelo, ya sea desde la base de datos o de un modelo base para testing.

El desacople de responsabilidades facilitó la lectura del código.

Además la misma clase Thermal implementaba SMP2 por lo tanto mezclaba lógica de la simulación y planificación en la clase encargada del cálculo. En la librería térmica la implementación del estándar quedo a cargo de la clase ThermalManager dejando al modelo la responsabilidad de manejar datos.

Parte de los refactorings realizados implicaron la clasificación de componentes del modelo, según su comportamiento térmico, abstrayéndose de sus capacidades eléctricas. Partiendo de la necesidad de que el simulador le asigne al modelo una lista de componentes de harness, se optó por separarlos según implementen la interface IDissipative_c o ISensor_c (caso contrario sería harness de superficies).

Para más detalle, ver diagrama de clases.

Diseño del Software de la librería ThermalCore

General

Esta sección provee la descomposición de la arquitectura de la LibThermalCore describiendo:

- Diferentes funciones y procesos
- Las relaciones entre las interfaces de los componentes
- Comportamiento dinámico del sistema

Las clases componentes de la LibThermalCore se dividen en dos grupos: aquellas que conforman el modelo y clases que realizan procesamiento.

Esta división lógica permite a futuro la inclusión de más modelos en simultáneo bajo una misma arquitectura de procesamiento.

Arquitectura Estática del Software detallada

Sistema Operativo

La librería se desarrolló en Ubuntu 12.04 64 bits, y será usada en igual sistema operativo.

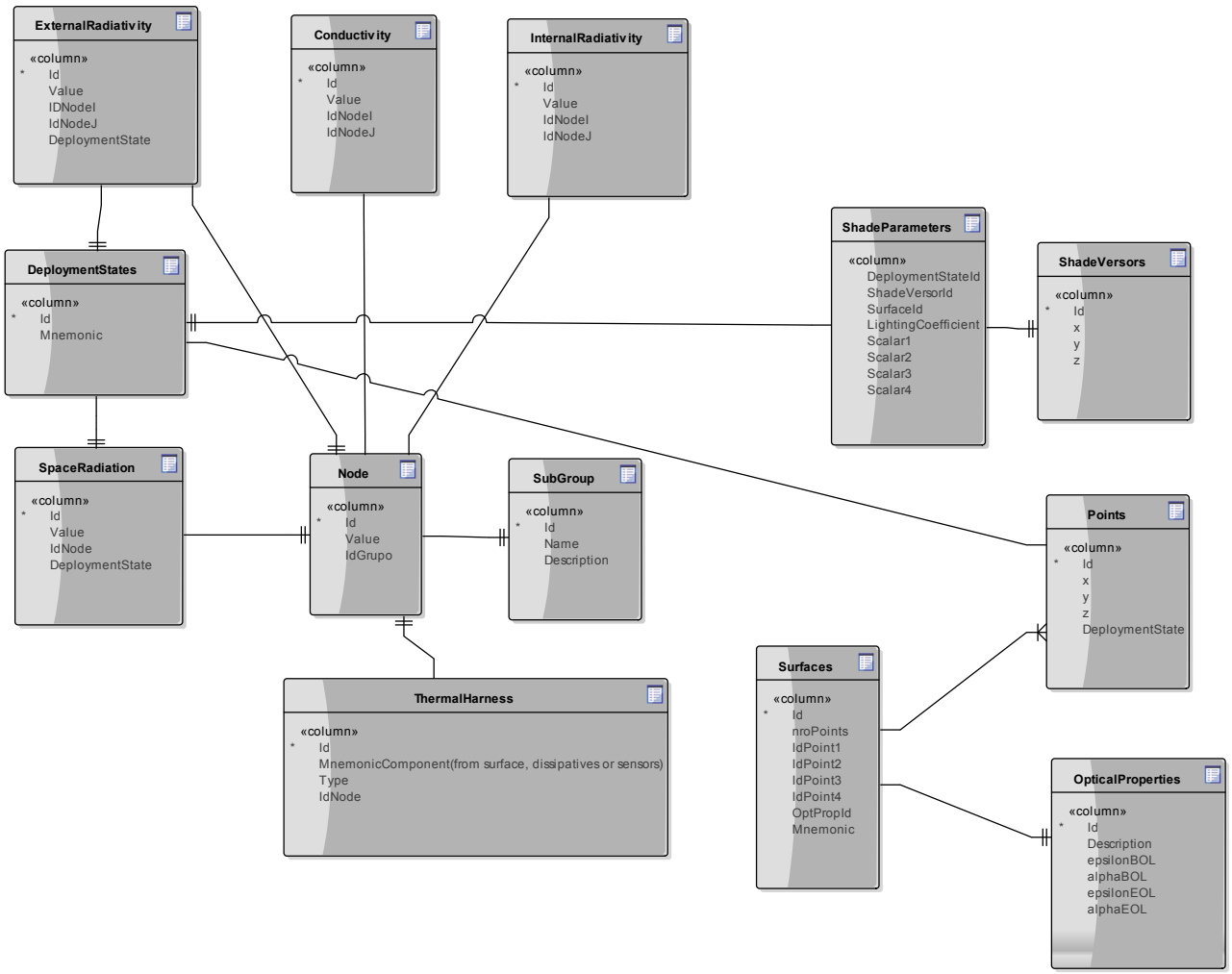
La arquitectura planteada hace uso de librerías compartidas que, para los sistemas Unix, se instalan en /usr/lib/local como ruta por defecto. Es necesario por lo tanto que parte de la instalación del STS valide la existencia de sus dependencias instaladas en el servidor.

Modelado de Datos de la ThermalCore

Los datos de la simulación térmica se encuentran distribuidos en dos bases de datos SQLite, ubicadas en las carpetas /src/data de cada proyecto (STS y ThermalCore).

De este modo en la base de datos "thermal" se ubicarán las tablas con información relacionada al cálculo térmico y en la "sts", información relacionada a componentes que pueden ser parte del harness en el modelo térmico reducido.

ThermalDB



61 Thermal BD

- Conductivity «table»

Lista las relaciones entre nodos, representando el factor con el que trasmiten el calor de uno al otro.

- DeploymentStates «table»

Estados de despliegue del satélite. Existen generalmente al menos 3 estados que representan al satélite: completamente plegado en situación de despegue, en proceso de despliegue, y totalmente desplegado que es la posición que adopta al llegar a la órbita.

- ExternalRadíativity «table»

Son las radiaciones entre nodos de las caras externas del satélite, varían según el estado de despliegue del mismo.

- InternalRadíativity «table»

Son las radiaciones entre nodos de las caras internas del satélite.

- Node «table»

Unidad mínima del modelo térmico, se les asigna un valor de capacitancia que representa la masa del nodo y la velocidad con que aumentará o disminuirá su temperatura (en iguales condiciones de temperatura un nodo con más masa tardará más tiempo en calentarse que uno de menor magnitud).

- OpticalProperties «table»

Las propiedades ópticas se utilizan en el modelo para diferenciar los materiales que forman parte del satélite, especificando el coeficiente épsilon de emisión y el alpha de absorción. Diferenciándolos a su vez tanto para el comienzo de la misión como para el final, debido a que es interesante poder realizar la simulación térmica teniendo en cuenta el paso de los años y el deterioro de los materiales componentes.

- Points «table»

Los puntos se utilizan para dar orientación en el espacio a las superficies del satélite, de esta manera será diferente la incidencia del Sol si una superficie se encuentra perpendicular o posicionada a 45° del Sol ya que en el primer caso recibirá el calor de manera más directa que en el otro y, por lo tanto, aumentará la temperatura más rápidamente.

- ShadeParameters «table»

Coeficientes de sombra para cada vector sombra de cada estado de despliegue.

- ShadeVersors «table»

Vectores de sombra que dependen del estado de despliegue.

- SpaceRadíation «table»

Las radiaciones espaciales representan la temperatura irradiada hacia el espacio, si este estuviera más frio que el satélite, tendería a enfriarlo. La temperatura del espacio es de 3°K.

- SubGroup «table»

Se utilizan los subgrupos para agrupar los nodos.

- Surfaces «table»

Las superficies se utilizan en el cálculo de la temperatura de los nodos ya que según sus propiedades y orientación modificarán en mayor medida a los nodos que se encuentren relacionados a ellas que a los que no.

- ThermalHarness «table»

El Thermal Harness representa la relación de cada nodo con algún componente que pueda afectar a su temperatura, ya sea una superficie, un sensor o un disipador.

Api Xml-Rpc

Es una interfaz que brindará acceso directo al modelo térmico. Permitiendo el uso de la API a los clientes. Se destinará por defecto el puerto servidor TCP 8890 donde atenderá las llamadas a procedimientos remotos.

Para implementar esta funcionalidad se dispone de las clases:

- XmlRpcServer_c

Esta clase inicia el servidor registrando todos los métodos de la API y creando un hilo de ejecución que queda escuchando en el puerto indicado anteriormente.

Ante la necesidad de poder ejecutar para testing un servidor dedicado a la funcionalidad térmica y otro dedicado a la funcionalidad de control de la simulación térmica. Se definieron los mismos en dos clases distintas:

- LibThermalXmlRpcServer_c

Esta clase registra al realizar su setUp, métodos de la API térmica definida en los archivos ThermalManagerXmlRpcMethods_c y ThermalModelXmlRpcMethods_c. Posee un método hook (addExtraMethods) que deben implementar las clases que hereden de ella para extender esta API.

- LibstsapiXmlRpcServer_c

Esta clase server hereda de la anterior, por lo que implementa en el método hook, el registro de los métodos propios del control de la simulación (aquellos definidos en SimulatorXmlRpcMethods_c). Como resultado final al instanciar este servidor, la clase cliente podrá acceder a la totalidad de métodos de la clase en cuestión y de su clase base.

- XmlRpcMethod_c

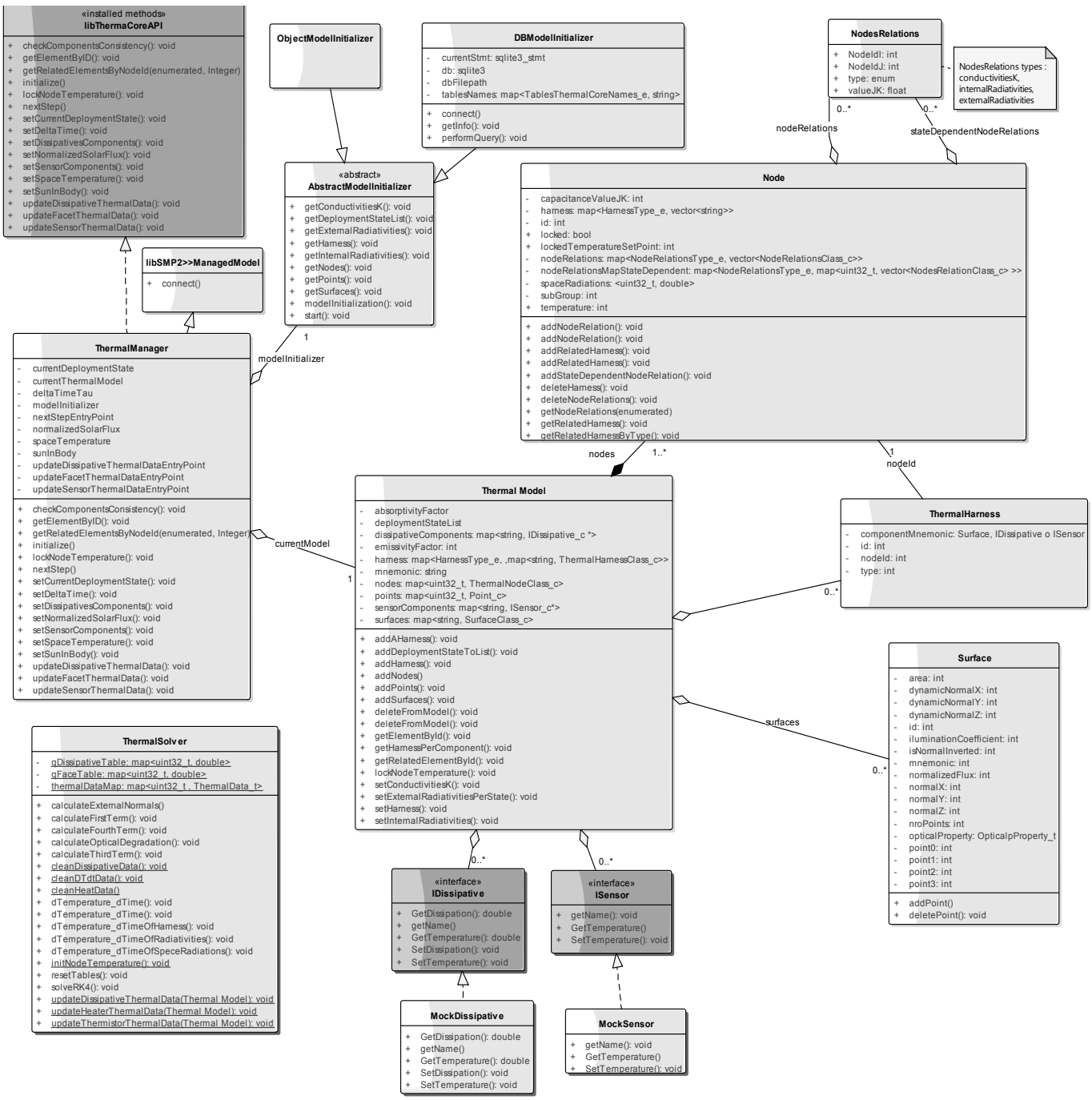
Los métodos rpc heredan de esta clase e implementan el método execute que se encarga de interpretar los parámetros, ejecutar el método de la API de la aplicación, y devolver un valor de resultado si correspondiese en formato xml. XmlRpcMethod_c hereda de xmlrpc_c::method de la librería xml-rpc -c su comportamiento.

Los métodos definidos en SimulatorXmlRpcMethods_c, ThermalModelXmlRpcMethods_c y ThermalManagerXmlRpcMethods_c heredan de XmlRpcMethod_c y redefinen el método execute y getName de la clase base.

- XmlRpcTools_c

Esta clase transforma la información obtenida desde la aplicación al formato xml que soporta la API.

Modelo Lógico LibThermalCore



62 Diagrama de clases LibThermalCore

- MockDissipative

Clase genérica que implementa IDissipative

- MockSensor

Clase genérica que implementa ISensor

- libSMP2>>ManagedModel

Clase Simulador base, que implementa SMP2

- ThermalManager

El manager coordina el cálculo térmico para determinado modelo. Es quien conoce cómo inicializar el modelo y la configuración de la simulación.

- Thermal Model

El modelo conoce quiénes lo componen y las reglas para agregar o eliminar nodos, relaciones y componentes. Un modelo está compuesto por nodos, relaciones entre ellos, componentes (Superficies, Sensores y Disipadores) y relaciones entre estos componentes y los nodos.

- Node

Unidad térmica significativa del modelo. Un nodo representa una parte del satélite. Posee cierta masa que hará que la temperatura del nodo varíe más rápida o lentamente. Puede estar conectado a otros nodos, superficies, sensores y disipadores, todos ellos son factores que influyen en el cálculo de la temperatura del nodo.

- NodesRelations

Las relaciones entre nodos, se modelaron para representar el factor de conductividad entre ellos es decir, con qué coeficiente se ponderará el calor al calcular cuánta temperatura de un nodo es trasmitida a otro.

- ThermalHarness

El harness modela la interacción de un nodo con algún sensor, disipador o superficie. Indicando un valor que represente cuánto afecta la presencia de ese componente en la temperatura del esa parte del satélite.

- ThermalSolver

El Solver es el encargado de implementar el cálculo diferencial para resolver la ecuación de calor del modelo. Utiliza un método para tal fin, que acumula los resultados preliminares en estructuras de datos y finalmente le asigna la temperatura acumulada correspondiente a un paso de la simulación a cada nodo.

- AbstractModelInitializer «abstract»

Esta clase abstracta define los métodos necesarios para la inicialización de un modelo térmico.

- DBModelInitializer

El inicializador a partir de una base de datos crea un modelo basado en los datos almacenados en la misma. La estructura de la base de datos debe seguir el diagrama indicado en la sección Modelado de Datos de la ThermalCore.

Luego de realizada la carga se efectúa automáticamente un chequeo de consistencia de los datos, de manera que las relaciones planteadas entre las partes tengan como precondición la existencia de las mismas.

- ObjectModelInitializer

El inicializador a partir de objetos crea un modelo base para testing del funcionamiento de la librería. El mismo consta de 3 nodos:

id = 0

-Relacionado con el nodo 1 con una conductividad de 1.13

-Relacionado con un disipador de 3 watts.

id = 1

-Relacionado con el nodo 2 con una conductividad de 1.13

-Con radiación espacial de 3.77119497111e-09

id = 2

-Relacionado con el nodo 0 con una conductividad de 1.13

-Relacionado con una superficie cuadrada con los puntos (0,0,0) ; (0,1,0); (0,0,1) y (1,0,0).

- Surface

Una superficie modela las caras de un satélite, puede presentar un porcentaje de sombra o puede estar totalmente iluminada, sus materiales pueden estar al final o al inicio de su ciclo de vida útil y absorber o emitir el calor de manera diferente en ambos casos. Puede ser una cara interna o externa dependiendo del estado de despliegue del satélite o puede quedar eclipsada por alguna otra pieza o la Tierra misma. Todas estas variantes son tenidas en cuenta también por el Solver para la definición de la temperatura de los nodos del satélite.

- IDissipative

Interfaz que agrupa a los componentes que disipan calor, los mismos puede ser heaters o diferentes tipos de cargas.

- ISensor

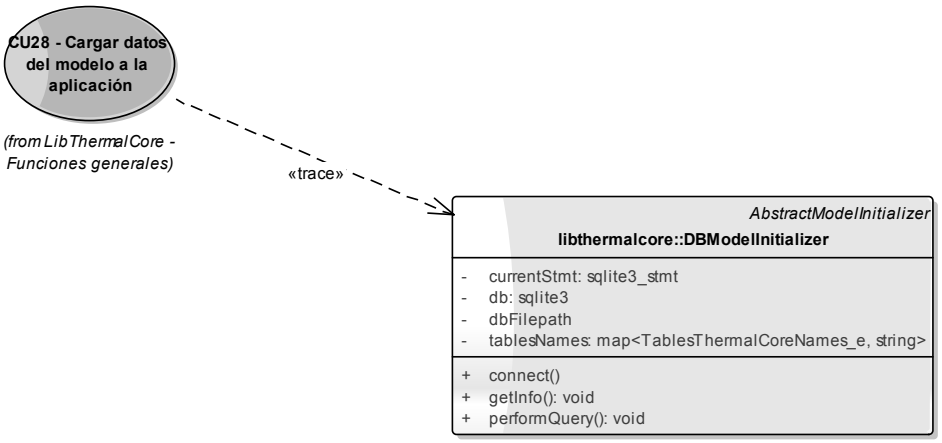
Interfaz que agrupa a los componentes que censan temperatura. Los mismos pueden ser termistores, termostatos o termocuplas.

- libThermaCoreAPI

API publicada por la LibThermalCore de manera que el server xml-rpc pueda registrar los métodos que la componen, encapsulando aquella funcionalidad que se espera no sea llamada por los clientes.

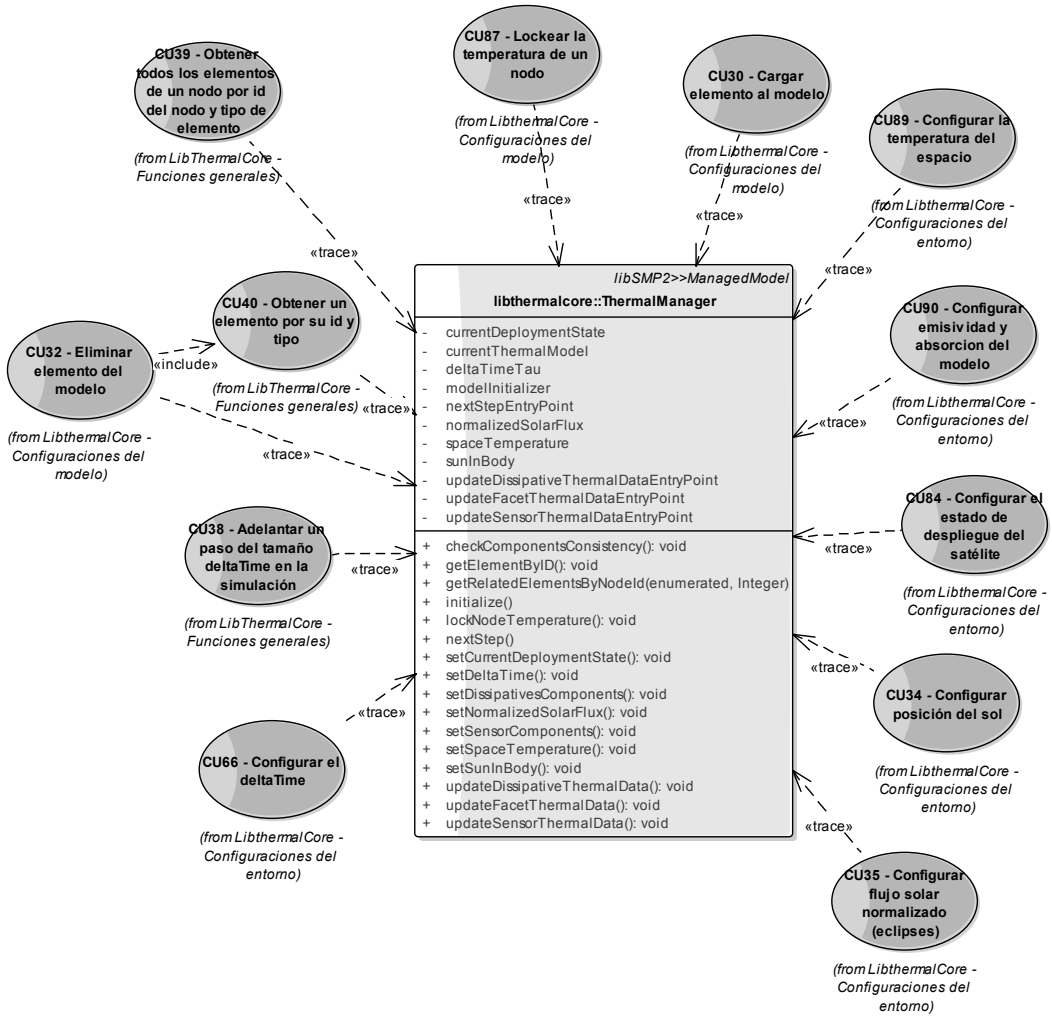
Implementación de Casos de Uso LibThermalCore

Casos de Uso de DBModelInitializer



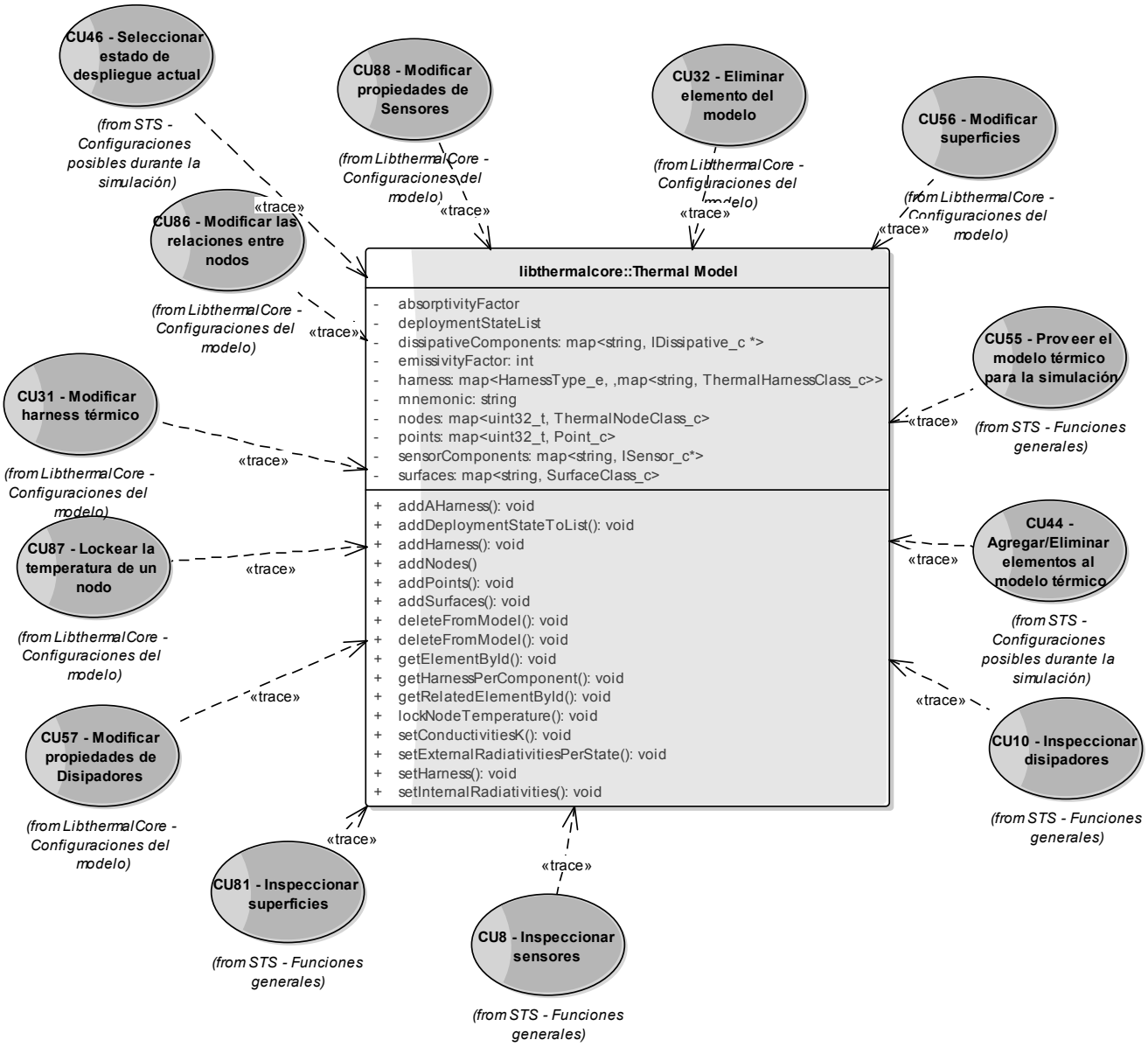
64 Casos de uso que implementa la clase Abstract Model Initializer

Casos de Uso de ThermalManager



63 Casos de uso que implementa la clase ThermalManager

Casos de Uso de ThermalModel



65 Casos de uso que implementa la clase ThermalModel

Capítulo 2: STS

Esta sección describe la arquitectura general desde un punto de vista de alto nivel.

General

El simulador térmico se encarga de representar la evolución de la temperatura de un satélite y sus diferentes partes.

Este desarrollo se diseñó usando SMP2 como base, con un diseño basado en clases.

El STS es una entidad capaz de recibir llamadas a procedimientos y de producir información sobre la simulación como resultado. Se utilizaron para esta funcionalidad las librerías xmlrpc-c y abyss-server.

Este SW puede operarse por medio de scripts, o mediante su interfaz gráfica. Ambas opciones de clientes, se comunican con la aplicación a través de la libthermalcoreclientapi y la libstsclientapi que proveen métodos tanto en C++ como en Python para el control y configuración de la simulación.

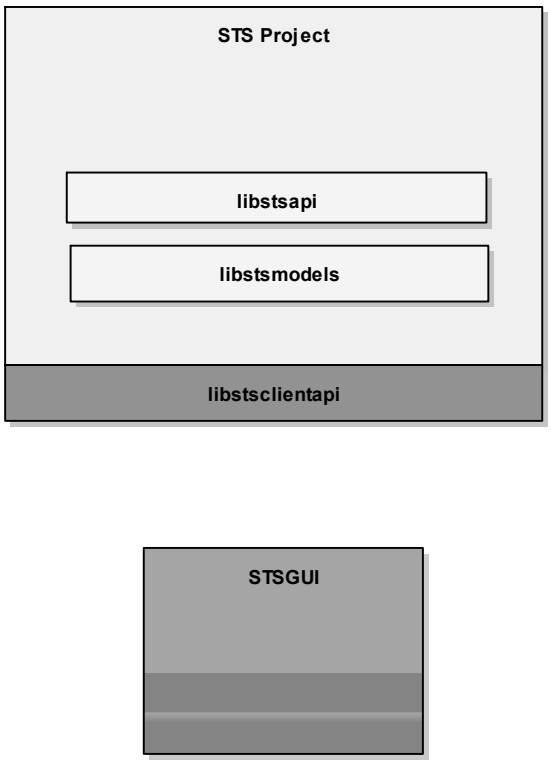
Gran parte de la funcionalidad del simulador es provista por la LibThermalCore, que provee el modelo térmico, pensada con el objetivo de reutilizar código que ya fue testeado.

Bajo esta arquitectura el cliente es el que inicia la comunicación solicitando al servidor que ejecute cierto procedimiento o función publicada en la API (libstsapi). Luego el servidor consulta el modelo térmico por medio de las funciones que publica la API de la LibThermalCore (libthermalcoreserverapi) y devuelve el resultado de dicha operación al cliente.

Las diferentes librerías con funcionalidad dedicada, permiten el desarrollo de una aplicación que separa los aspectos específicos de la plataforma de los aspectos específicos del modelo, obteniendo así portabilidad.

Arquitectura Estática del Software

Estructura de la implementación del STS



66 Estructura la implementación del del STS

- STSGUI

Interfaz gráfica que, haciendo uso de la funcionalidad de la libthermalclientapi y la libstsclientapi, ofrece de manera más intuitiva la posibilidad de controlar la simulación térmica del modelo a través del tiempo. Permite la inspección de los nodos, disipadores, sensores, superficies y relaciones entre nodos y de harness. Así mismo la temperatura de los nodos puede verse graficada y es posible realizar modificaciones en el modelo a simular, para diseñar mejorías en el mismo, antes de persistirlo.

Se implementó con PyQT4 y se diseñó haciendo uso del QTCreator.

- libstsclientapi

Esta librería ofrece la posibilidad de comunicarse con el servidor xmlrpc correspondiente.

Puede utilizarse la librería tanto en C++ como en Python, debido a que se utilizó la herramienta SWIG.

Una vez implementada esta librería, se realizó testing funcional del STS por medio de scripts python que, en cada método recorrían de manera transversal la arquitectura de librerías desarrolladas: libstsclientapi -libstsapi - sts-libthermacore

- STS Project

Aplicación que corre en Linux 64 bits, como proceso background. Simula la temperatura satelital en base a un modelo.

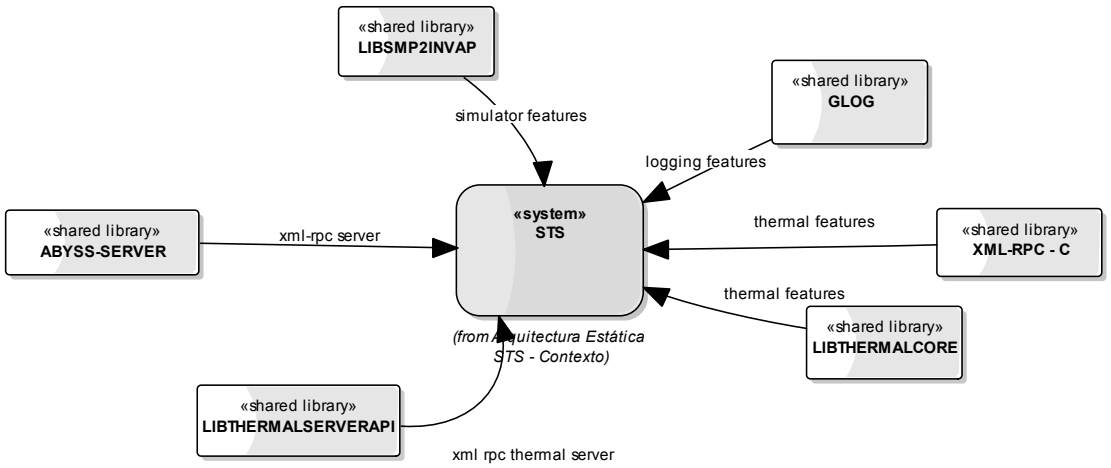
- libstsmodels

Esta librería publica ciertas clases que exponen funcionalidad para configurar la simulación en cuanto a su velocidad, tamaño del paso de la simulación, controles de los estados de la misma, consultar y modificar tiempos.

- libstsapi

Esta librería ofrece un server xml-rpc, que facilitan la comunicación e interacción de los clientes con el STS. Este servidor registra los métodos que publica la libstsmodels

Contexto de la implementación del STS



67 Diagrama de contexto del STS

- ABYSS-SERVER «shared library»

Ya descripto.

- GLOG «shared library»

Ya descripto.

- LIBSMP2 «shared library»

Provee la funcionalidad de modelos y servicioes de SMP2. Estos modelos pueden contener tareas que se agregan en el scheduler del simulador.

- LIBTHERMALCORE «shared library»

Provee funcionalidad térmica a los simuladores de satélite

- LIBTHERMALSERVERAPI «shared library»

Crea un servidor xml rpc que registra los métodos expuestos por la LibThermalCore

- XML-RPC – C «shared library»

Ya descripto.

Funcionalidad

El STS cubre las siguientes funciones según se especifican en el SRS.

Un Operador puede:

- Inspeccionar los distintos elementos del modelo del satélite, conociendo su estado y propiedades a través de la interfaz gráfica.
- Configurar la velocidad de la simulación, la posición del Sol respecto del satélite, el flujo solar normalizado, seleccionar estado de despliegue, configurar el tiempo que se representará en cada paso de la simulación.
- Modificar parte del modelo: sensores, disipadores, superficies, harness y relaciones entre nodos.
- Lockear la temperatura de un nodo en la simulación.

Se puede proveer por medio de llamadas a la API, información de ambiente a la aplicación STS.

El operador Tester puede realizar operaciones de consulta sobre la simulación a través de métodos wrappeados a Python de la libthermalcoreserverapi y la libstsapi.

Estándares de diseño, convenciones y procedimientos

General

El proceso aplicado en el ciclo de vida del desarrollo de Software será acorde a lo establecido en el documento de Proceso de Desarrollo de SW del servicio de Software de Invap.

Método de diseño de la arquitectura de Software

La metodología usada para el desarrollo del STS se basó en COMET al igual que la LibThermalCore.

Patrón de Arquitectura: Model View Controller

Este patrón arquitectónico define que la solución debe de dividirse en tres capas lógicas: presentación, lógica de negocio y acceso a datos, con esto se busca especificar el conjunto de responsabilidades de cada una de las capas y los componentes que los conformarán.

En la implementación del STS se pueden observar:

- Presentación:

La clase STS_c hace de interfaz para la librería.

- Lógica de Negocio:

La clase Simulator (base de STS) implementa la funcionalidad de la lógica de negocio.

- Acceso a Datos:

El DBInitializer es quien sabe leer la información desde la base de datos y crear objetos de tipo LoadContinuedCurrentAndContinuedPotencyClass_c, LoadContinuedResistanceClass_c, ThermistorClass_c, ThermostatClass_c, etc.

Estándares de documentación de código fuente

El Software STS se modelo usando Unified Model Language (UML).

Estándares de implementación de simuladores: SMP2

Se implementa el estándar SMP2 (Simulation Model Portability 2) que propone una arquitectura modular para el desarrollo de simuladores portables. La misma cubre dos tipos de componentes: la Simulación con instancias de Modelos que proveen el comportamiento específico a la aplicación, y el Ambiente de Simulación que provee los servicios de simulación. Esta arquitectura se encuentra implementada en la librería libsmp2.

Diseño del Software del STS

General

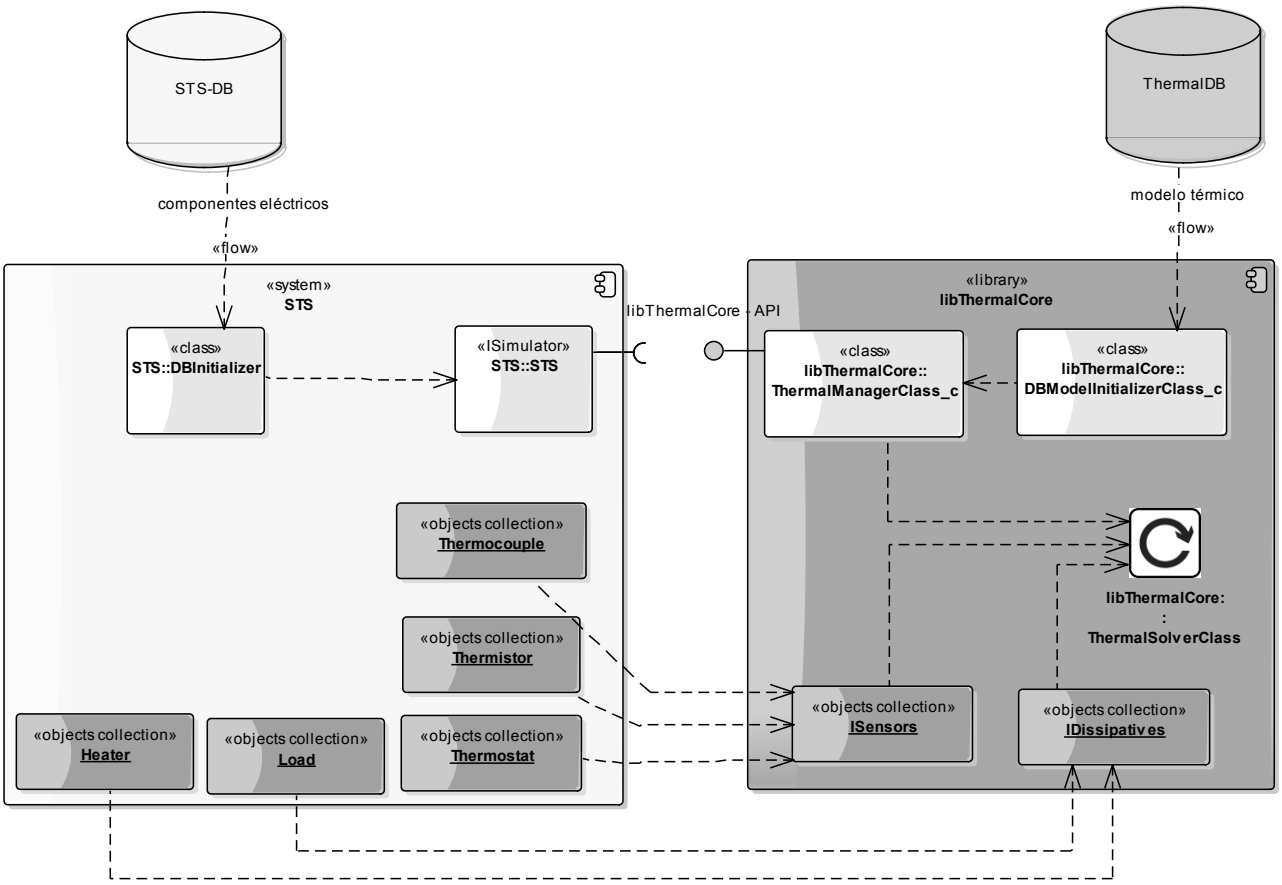
Esta sección provee la descomposición de la arquitectura del STS describiendo:

- Diferentes funciones y procesos
- Las relaciones entre las interfaces de los componentes
- Comportamiento dinámico del sistema

Las clases componentes del STS se dividen en dos grupos: aquellas que realizan la carga del modelo y clases que realizan la simulación que son las provistas por la librería libsmp2.

Arquitectura Estática del Software detallada

Componentes del STS



68 Diagrama de componentes del STS

- STS «system»
  - DBInitializer «class»

    Hace las veces de ORM. Su funcionalidad es leer los datos de la base de datos del STS y generar los objetos que representen esos datos.
  - STS «ISimulator»

    Es la clase que implementa el comportamiento del simulador propiamente dicho.
  - Heater «objects collection»

    Heaters que forman parte del satélite. Un heater convierte la electricidad en calor por el proceso de calentamiento resistivo.
  - Load «objects collection»

    Cargas que forman parte del satélite. Pueden ser de corriente continua, de potencia continua o de resistencia continua.
  - Thermistor «objects collection»

    Termistores que forman parte del satélite. Es un sensor de temperatura. Su funcionamiento se basa en la variación de la resistividad que presenta un semiconductor con la temperatura.

- Thermocouple «objects collection»

  Thermocuplas que forman parte del satélite. Es un sensor que al aplicarle temperatura genera voltaje.

- Thermostat «objects collection»

  Termostatos que forman parte del satélite. Un termostato es el componente de un sistema de control simple que abre o cierra un circuito eléctrico en función de la temperatura.

- STS-DB «repository»

Contiene información de los componentes eléctricos que conforman el satélite.

- LibThermalCore «library»

- DBModelInitializerClass_c «class»

Hace las veces de ORM. Su funcionalidad es leer los datos de la base de datos térmica y generar los objetos que representen esos datos.

- ThermalManagerClass_c «class»

Se encarga de coordinar el funcionamiento de la LibThermalCore, exponiendo una API, proveyéndole el modelo térmico, sensores y disipadores al ThermalSolver.

- ThermalSolverClass «class»

Se encarga de realizar los cálculos necesarios para avanzar un paso en la simulación térmica. Usando un método de cálculo y teniendo en cuenta la posibilidad de lockear la temperatura de un nodo en una temperatura en particular

- IDissipatives «objects collection»

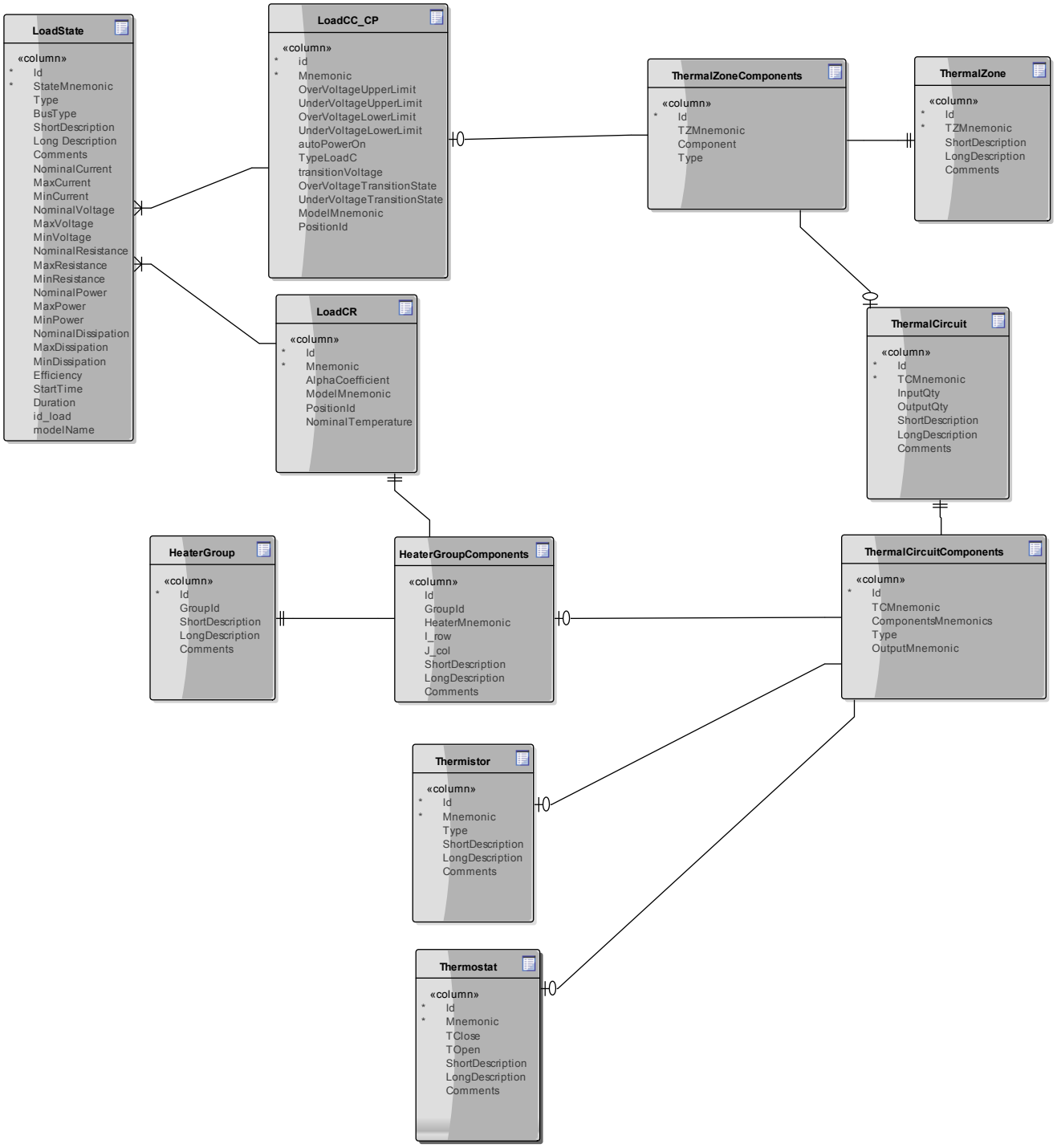
Es una colección de objetos que conocen la potencia que disipan, su temperatura instantánea y nombre. Se consultan estos valores durante el cálculo del siguiente paso por parte del ThermalSolver.

- ISensors «objects collection»

Es una colección de objetos que conocen su temperatura instantánea y nombre. Se consultan estos valores durante el cálculo del siguiente paso por parte del ThermalSolver.

Modelado de Datos del STS

StsDB



69 STSDB

- HeaterGroup «table»

Define un conjunto que de componentes que genera calor en el satélite.

- HeaterGroupComponents «table»

Enumera los componentes que forman un grupo de heaters, estos pueden ser: cargas de resistencia continua.

- LoadCC_CP «table»

Cargas de corriente o potencia continua del satélite. Una carga puede tener diferentes estados.

- LoadCR «table»

Cargas de resistencia continua del satélite. Una carga puede tener diferentes estados.

- LoadState «table»

Se muestran los distintos estados que puede presentar una carga mostrando los umbrales de potencia, voltaje, disipación, corriente, resistencia y temperatura que puede adquirir en cada caso.

- ThermalCircuit «table»

Define un circuito térmico en el satélite.

- ThermalCircuitComponents «table»

Enumera los componentes que forman un circuito térmico: pueden ser grupos de heater, termistores o termostatos.

- ThermalZone «table»

Define una zona térmica en el satélite.

- ThermalZoneComponents «table»

Enumera los componentes que forman una zona térmica: pueden ser cargas de corriente o potencia continua o circuitos térmicos.

- Thermistor «table»

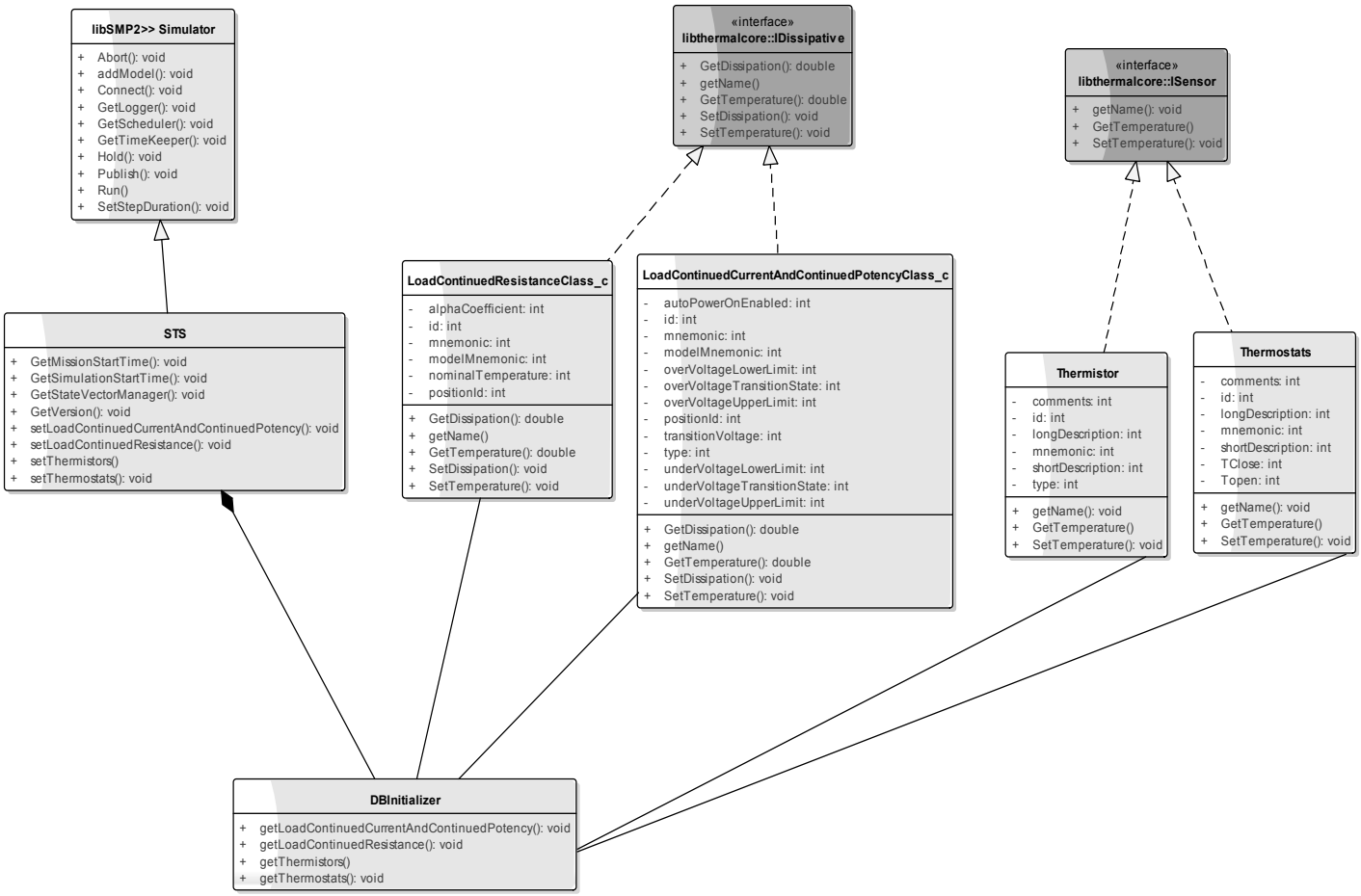
Termistores del satélite.

- Thermostat «table»

Termostostatos del satélite.

Libstsmodels

Modelo Lógico libstsmodels



70 Diagrama de clases LibStsModels

- DBInitializer
  Clase encargada de leer la base de datos y generar objetos para cargar al STS

- LoadContinuedCurrentAndContinuedPotencyClass_c
  Subsistemas o componentes de potencia y corriente continuas

- LoadContinuedResistanceClass_c

Subsistemas o componentes de resistencia continua

- STS

Simulador de la evolución térmica del satélite,

- Thermistor

Un termistor es un tipo de resistor cuya resistencia es dependiente de la temperatura

- Thermostats

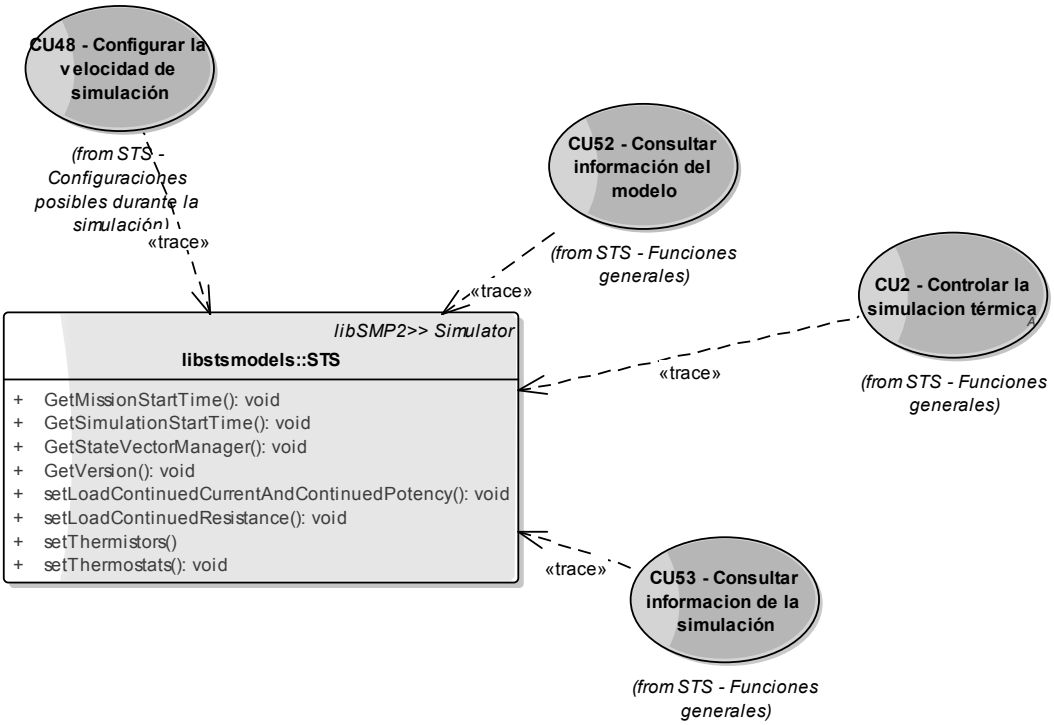
Un termostato es un componente de los cuales detecta la temperatura de un sistema de modo que la temperatura del sistema se mantiene cerca de un punto de ajuste deseado .

- libSMP2>> Simulator

Clase Simulator base que implementa servicios SMP2

Implementación de Casos de Uso STS

Casos de Uso STS



71 Casos de uso que implementa la clase STS

Apéndice C – Procedimiento de Testing

Introducción

Propósito

Este documento define los procedimientos de prueba que acompañan a cada Tag del Software de la aplicación STS.

Alcance

Este documento contiene test cases que verifican casos de éxito de las principales funcionalidades del STs.

Equipamiento Requerido

Perfil y cantidad de personas

Cantidad de personas: 1

Perfil: Tester

Software

La instalación y configuración del STS están descriptas en Manual de Usuario.

Hardware

PC compatibles Intel I7 con 8 GB de Memoria RAM y HDD de 500 GB.

Recomendado: Server Intel Xeon con 12 cores, 16 Gb de Ram y HDD de 500 GB

Sistema Operativo Ubuntu v.12.04 actualizado, 64 bits.

Placa de Red de 100 Mbps.

Monitor LCD 20''

Tiempo estimado de ejecución

Cantidad de horas: 1

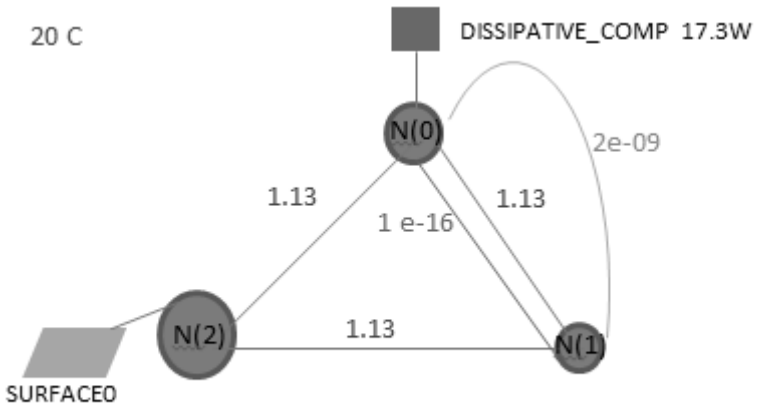
Especificación del procedimiento de prueba

TP-STS-ThermalModelCalculation -01

Propósito

El propósito de este procedimiento es verificar configuraciones y controles del STS a través de la STSGUI, en base a un modelo básico conocido.

Modelo Básico



72 Modelo térmico mock para testing

Configuración inicial

En el archivo STS/src/stsEnv.sh modificar las variables de entorno:

a) THERMAL_DATABASE_PATH: con la ubicación absoluta de la base de datos térmica
b) THERMAL_MODEL_SOURCE="OBJECT_INITIALIZER": para levantar el modelo térmico básico para realizar testing.

Ejecutar el script STS/src/./run

Ejecutar el script STSGUI/src/./run

Valores por defecto de parámetros de contexto

Parámetro	Valor
Space Temperature	3 K
Delta Time	0.2 ms
NormalizedSolarFlux	1
Sun in body	(1, 0, 0)
Deployment State	0
Emissivity	0
Absorptivity	0
Step Duration	10 ms
SpeedFactor	1
Temperatura inicial de nodos	20°C
Temperatura inicial de disipadores	0°C
Temperatura inicial de sensores	0°C

73 Valores por defecto STS

Pasos

1 Verificar los valores por defecto en la STSGUI, según 0.

2 Ejecutar TC-Consistent Model-01S

3 Apagar el STS para reiniciar el modelo
4 Ejecutar el script STS/src/./run
5 Verificar los valores por defecto en la STSGUI, según 0.
6 Ejecutar TC-Space Temperature-01S

7 Apagar el STS para reiniciar el modelo
8 Ejecutar el script STS/src/./run
9 Verificar los valores por defecto en la STSGUI, según 0.
10 Ejecutar TC-Component Dissipation-01S

11 Apagar el STS para reiniciar el modelo
12 Ejecutar el script STS/src/./run
13 Verificar los valores por defecto en la STSGUI, según 0.
14 Ejecutar TC-Sensor Component-01S

15 Apagar el STS para reiniciar el modelo
16 Ejecutar el script STS/src/./run
17 Verificar los valores por defecto en la STSGUI, según 0.
18 Ejecutar TC-Conductivity Relation-01S

19 Apagar el STS para reiniciar el modelo
20 Ejecutar el script STS/src/./run
21 Verificar los valores por defecto en la STSGUI, según 0.
22 Ejecutar TC-External Radíativities-01S

23 Apagar el STS para reiniciar el modelo
24 Ejecutar el script STS/src/./run
25 Verificar los valores por defecto en la STSGUI, según 0
26 Ejecutar TC-Nodes Relations Management-01S

27 Apagar el STS para reiniciar el modelo
28 Ejecutar el script STS/src/./run
29 Verificar los valores por defecto en la STSGUI, según 0.
30 Ejecutar TC-Sun In Body -01S

31 Apagar el STS para reiniciar el modelo
32 Ejecutar el script STS/src/./run
33 Verificar los valores por defecto en la STSGUI, según 0.
34 Ejecutar TC-Lock Temperature -01S

35 Apagar el STS para reiniciar el modelo
36 Ejecutar el script STS/src/./run
37 Verificar los valores por defecto en la STSGUI, según 0.
38 Ejecutar TC-Surfaces Management-01S

39 Apagar el STS para reiniciar el modelo
40 Ejecutar el script STS/src/./run
41 Verificar los valores por defecto en la STSGUI, según 0.
42 Ejecutar TC-Dissipatives Management-01S

Especificación de Test Cases

TC-Consistent Model-01S

Resumen: El propósito de este test case es verificar que la temperatura de los nodos sin harness, tienden a la temperatura del espacio cuando es menor		
Precondiciones: Servidor y STSGUI se encuentran corriendo. Verificar los valores por defecto en la STSGUI.		
#	**Acciones**:	**Resultados Esperados**:
1	En la ventana principal setear normalized solar flux = 0 y confirmar	Verificar que el combo de normalized solar flux = 0
2	Setear speed factor = x4	Verificar que el speed running alcanza la velocidad 4
3	Presionar el boton “View Thermal Model”	Verificar que se abrió la herramienta de inspección del modelo térmico en una nueva ventana
4	Ir a la pestaña de dissipatives, seleccionar DISSIPATIVE_COMP y presionar “delete Harness”	Verificar que en la pestaña dissipatives, fila “DISSIPATIVE_COMP”, columna “Nodes Ids” no existe una relación de harness (“-”)
5	Ir a la pestaña de surfaces, seleccionar SURFACE0 y presionar “delete Harness”	Verificar que en la pestaña surfaces, fila “SURFACE0”, columna “Nodes Ids” no existe una relación de harness (“-”)
6	En la ventana principal presionar play	Verificar que el estado del simulador es “Executing”
7	Esperar 20 horas de simulación	Verificar que la temperatura de los 3 nodos se asemeja y se acerca a los -270.15 °C

TC-Space Temperature – 01S

Resumen: El propósito de este test case es verificar que la temperatura de los nodos sin harness, tienden a la temperatura del espacio cuando esta aumenta		
Precondiciones: Servidor y STSGUI se encuentran corriendo. Verificar los valores por defecto en la STSGUI.		
#	**Acciones**:	**Resultados Esperados**:
1	En la ventana principal setear normalized solar flux = 0 y confirmar	Verificar que el combo de normalized solar flux = 0
2	Setear speed factor = x4	Verificar que el speed running alcanza la velocidad 4
3	Setear space temperature = 300K (27°C) y confirmar	Verificar que el combo muestra 300 K
4	Presionar el boton “View Thermal Model”	Verificar que se abrió la herramienta de inspección del modelo térmico en una nueva ventana
5	Ir a la pestaña de dissipatives, seleccionar DISSIPATIVE_COMP y presionar “delete Harness”	Verificar que en la pestaña dissipatives, fila “DISSIPATIVE_COMP”, columna “Nodes Ids” no existe una relación de harness (“-”)
6	Ir a la pestaña de surfaces, seleccionar SURFACE0 y presionar “delete Harness”	Verificar que en la pestaña surfaces, fila “SURFACE0”, columna “Nodes Ids” no existe una relación de harness (“-”)
7	En la ventana principal presionar play	Verificar que el estado del simulador es “Executing”
8	Esperar 3 minutos de simulación	Verificar que la temperatura de los 3 nodos aumenta

TC-Component Dissipation-01S

Resumen: El propósito de este test case es verificar que la temperatura de un nodo aumenta al ponerle un disipador de potencia.		
Precondiciones: Servidor y STSGUI se encuentran corriendo. Verificar los valores por defecto en la STSGUI.		
#	**Acciones**:	**Resultados Esperados**:
1	En la ventana principal setear normalized solar flux = 0 y confirmar	Verificar que el combo de normalized solar flux = 0
2	Setear speed factor = x4	Verificar que el speed running alcanza la velocidad 4

3	Presionar el boton “View Thermal Model”	Verificar que se abrió la herramienta de inspección del modelo térmico en una nueva ventana
4	Ir a la pestaña de dissipatives, seleccionar DISSIPATIVE_COMP y presionar “delete Harness”	Verificar que en la pestaña dissipatives, fila “DISSIPATIVE_COMP”, columna “Nodes Ids” no existe una relación de harness (“-”)
5	Ir a la pestaña “Nodes”, seleccionar cada fila en la tabla y presionar por cada una “Add curve”	Verificar que en la sección de gráfico aparecen los ids de los nodos
6	En la ventana principal presionar play	Verificar que el estado del simulador es “Executing” y que comienza a graficarse la temperatura de los 3 nodos
7	Esperar 2 minutos de simulación	Verificar que la temperatura de los 3 nodos disminuye
8	Pausar simulación	Verificar que el estado es Standby y que se mantiene el tiempo de simulación
9	Ir a la pestaña “Dissipatives”, seleccionar “DISSIPATIVE_COMP”, y presionar “Add as Harness”	Verificar que una ventana de diálogo se abre con un combo que lista los ids de los 3 nodos.
10	Elegir el id 0 y presionar “Ok”	La ventana de diálogo se cierra y aparece en la columna “Nodes Ids” el id 0
11	Presionar Play	Verificar estado “Executing”
12	Esperar 3 minutosde simulación	Verificar que la temperatura del nodo 0 aumenta y que la curva del nodo 0 crece.

TC-Sensor Component -01S

Resumen: El propósito de este test case es verificar que un sensor lee la temperatura del nodo con el que se relaciona		
Precondiciones: Servidor y STSGUI se encuentran corriendo. Verificar los valores por defecto en la STSGUI.		
#	**Acciones**:	**Resultados Esperados**:
1	En la ventana principal setear normalized solar flux = 0 y confirmar	Verificar que el combo de normalized solar flux = 0
2	Setear speed factor = x4	Verificar que el speed running alcanza la velocidad 4
3	Presionar el boton “View Thermal Model”	Verificar que se abrió la herramienta de inspección del modelo térmico en una nueva ventana
4	Seleccionar uno a uno los nodos y por cada uno presionar “Add curve”	Verificar que aparecen los ids en la sección de grafico
5	Presionar Play	Verificar que el estado de simulación es “Executing”
6	Esperar 2 minutosde simulación	Verificar que todos los nodos se enfrían
7	Pausar la simulación	Verificar que el estado de simulación es “Standby” y se mantiene el tiempo de simulación
8	Ir a la pestaña de “Sensor” y presionar “New Sensor”	Verificar que se abre una nueva ventana con un input para el nombre del sensor y otro para la temperatura
9	Completar nombre = “SENSOR0” y temperature = 68.3°C y presionar “Ok”	Verificar que aparece una fila nueva en la tabla que muestra los nuevos datos
10	Seleccionar “SENSOR0” y presionar “Add as harness”	Verificar que se abre una ventana con un combo que lista los nodos
11	Seleccionar el nodo 1 y presionar “Ok”	Verificar que aparece el id 1 en la columna de nodes ids
12	Presionar play	Verificar que el estado es “Executing”
13	Esperar 3 minutosde simulación	Verificar que el sensor toma la temperatura del nodo

TC-Conductivity Relation-01S

Resumen: El propósito de este test case es verificar que una relación de conductividad mayor transmite más rápido la temperatura de un nodo a otro.		
Precondiciones: Servidor y STSGUI se encuentran corriendo. Verificar los valores por defecto en la STSGUI.		
#	**Acciones**:	**Resultados Esperados**:
1	En la ventana principal setear normalized solar flux = 0 y confirmar	Verificar que el combo de normalized solar flux = 0
2	Setear speed factor = x4	Verificar que el speed running alcanza la velocidad 4
3	Presionar el boton “View Thermal Model”	Verificar que se abrió la herramienta de inspección del modelo térmico en una nueva ventana
4	Seleccionar cada fila en la tabla y presionar por cada una “Add curve”	Verificar que en la sección de gráfico aparecen los ids de los nodos
5	En la ventana principal presionar play	Verificar que el estado del simulador es “Executing” y que comienza a graficarse la temperatura de los 3 nodos
6	Esperar 2 minutosde simulación	Verificar que la temperatura de los 3 nodos disminuye
7	Pausar la simulación	Verificar que el estado es Standby y se mantiene el tiempo de simulación
8	Ir a la pestaña “Nodes Relations”, seleccionar el par (0,1) y presionar “Delete Conductivity”	Verificar que en la tabla, no existe una relación de conductividad para el par (0,1)
9	Presionar “Add Conductivity”	La ventana de diálogo se abre mostrando dos combos con los nodos disponibles y un campo editable para completar con el valor de la relación
10	Elegir nodo 0, nodo 1 y valor 8 y presionar “Ok”	Verificar que se crea la relación.
11	Presionar Play	Verificar que el estado es “Executing”
12	Esperar 30 segundos de simulación	Verificar que la temperatura del nodo 0 decae más rápidamente que en los segundos anteriores

TC-External Radiativities – 01S

Resumen: El propósito de este test case es verificar que en la pestaña “Nodes Relations” se listan las relaciones externas del estado de despliegue actual.		
Precondiciones: Servidor y STSGUI se encuentran corriendo. Verificar los valores por defecto en la STSGUI.		
#	**Acciones**:	**Resultados Esperados**:
1	Presionar el boton “View Thermal Model”	Verificar que se abrió la herramienta de inspección del modelo térmico en una nueva ventana
2	Ir a la pestaña “Nodes Relations”	Verificar que en la tabla, existe una relación de radiatividad externa para el par (0,1)
3	En la ventana principal cambiar el estado de despliegue a 4	Verificar que el combo muestra 4-fully deployed y que en la columna External Radiativities del par (0, 1) no aparecen relaciones.
4	En la ventana principal cambiar el estado de despliegue a 0	Verificar que el combo muestra 4-fully stowed y que en la columna External Radiativities del par (0, 1) aparece nuevamente la relación.

TC-Nodes Relations Management-01S

Resumen: El propósito de este test case es verificar que es posible listar los pares de nodos relacionados sin repetirse entre sí.		
Precondiciones: Servidor y STSGUI se encuentran corriendo. Verificar los valores por defecto en la STSGUI.		
#	**Acciones**:	**Resultados Esperados**:
1	Presionar el boton “View Thermal Model”	Verificar que se abrió la herramienta de inspección del modelo térmico en una nueva ventana
2	Ir a la pestaña “Nodes Relations”	Verificar que en la tabla, existe una relación de conductividad para el par (0,2)
3	Seleccionar el par (0,2) y presionar “Delete Conductivity” y confirmar	Verificar que el par desaparece de la tabla
4	Presionar “Add Internal Radíativity”	Verificar que se abre una nueva ventana con dos combos que listan los nodos y un input para completar con el valor de la relación
5	Elegir los nodos 0 y 2 y completar con valore = 620 y presionar “Ok”	Verificar que se añade una nueva fila a la tabla con el par (0, 2) y el valor 620 en la columna de “Internal Radiativities”

TC-Sun In Body-01S

Resumen: El propósito de este test case es verificar que la incidencia del sol en superficies relacionadas a nodos, aumenta la temperatura de los mismos		
Precondiciones: Servidor y STSGUI se encuentran corriendo. Verificar los valores por defecto en la STSGUI.		
#	**Acciones**:	**Resultados Esperados**:
1	Presionar el boton “View Thermal Model”	Verificar que se abrió la herramienta de inspección del modelo térmico en una nueva ventana
2	Seleccionar el nodo2 y presionar “AddCurve”	Verificar que aparece el Id 2 en la sección de gráfico
3	Ir a la pestaña “Surfaces”, seleccionar “SURFACE0” y presionar “Delete Harness”	Verificar que se eliminar el id 0 de la columna “Nodes Ids”
4	Presionar Play	Verificar que el estado de simulación es “Executing”
5	Esperar 30 segundos de simulación	Verificar que la temperatura del nodo decrece
6	Pausar la simulación	Verificar que el estado de simulación es “Standby” y se mantiene el tiempo de simulación
7	Ir a la pestaña “Surfaces”, seleccionar “SURFACE0” y presionar “Add as harness”	Verificar que se abre una ventana nueva con un combo que muestra los ids de los nodos
8	Seleccionar nodo2 y presionar “Ok”	Verificar que se lista el id 2 en la columna “Nodes Ids”
9	Presionar Play	Verificar que se reanuda la simulación y el estado es “Executing”
10	Esperar 30 segundos de simulación	Verificar que la temperatura del nodo 2 aumenta en comparación a paso 4
11	Pausar la simulación	Verificar que el estado de simulación es “Standby” y se mantiene el tiempo de simulación
12	Modificar la posición del sol a (2, 0, 4) y confirmar el cambio	El combo de sun in body debe mostrar (2, 0, 4)
13	Presionar Play	Verificar que se reanuda la simulación y el estado es “Executing”
14	Esperar 30 segundos de simulación	Verificar que la temperatura del nodo 2 disminuye en comparación a paso 9

TC-Lock Temperature-01S

Resumen: El propósito de este test case es verificar que se puede configurar la simulación para que detenga la temperatura del nodo en un valor predeterminado		
Precondiciones: Servidor y STSGUI se encuentran corriendo. Verificar los valores por defecto en la STSGUI.		
#	**Acciones**:	**Resultados Esperados**:
1	Presionar el boton “View Thermal Model”	Verificar que se abrió la herramienta de inspección del modelo térmico en una nueva ventana
2	Seleccionar el nodo2 y presionar “Lock Temperature”	Verificar que aparece una ventana de diálogo
3	Completar con el valor 33°C y presionar “Ok”	Verificar que la ventana de diálogo se cierra, se colorea de violeta la celda de la temperatura del nodo2, y se completa con 33°C la columna “Lock Temperature” para el nodo2
4	Presionar Play	Verificar que el estado de simulación es “Executing”
5	Esperar 30 segundos de simulación	Verificar que la temperatura del nodo2 crece y se detiene en 33°C mientras los demás nodos continúan con su variación térmica
6	Seleccionar el nodo2 y presionar “Lock Temperature”	Verificar que se abre una ventana de diálogo o que pregunta si se quiere desbloquear la temperatura del nodo
7	Confirmar	Verificar que se vuelve a blanco la celda de la temperatura del nodo y que se elimina de la columna “Locked Temperature” el 33°C. Además la temperatura del nodo deberá disminuir.

TC-Surfaces Management -01S

Resumen: El propósito de este test case es verificar que se puede crear y eliminar una superficie		
Precondiciones: Servidor y STSGUI se encuentran corriendo. Verificar los valores por defecto en la STSGUI.		
#	**Acciones**:	**Resultados Esperados**:
1	Presionar el boton “View Thermal Model”	Verificar que se abrió la herramienta de inspección del modelo térmico en una nueva ventana
2	Ir a la pestaña de “Surfaces” , seleccionar “SURFACE0” y presionar “Delete Surface”	Verificar que aparece un cartel de error que indica que está relacionada a un nodo
3	Presionar “Delete Harness", y confirmar	Verificar que se elimina el nodo2 de la columna “Nodes Ids”
4	Presionar “Delete Surface” y confirmar	Verificar que se elimina la fila de la tabla quedando la tabla vacía
5	Presionar “New Surface”	Verificar que se abre una nueva ventana con distintas propiedades de una superficie
6	Completar nombre “SURFACE1” , seleccionar 2 puntos y presionar “Ok”	Verificar que se lista “SURFACE1” en la tabla con dos puntos en la columna “Points”

TC-Dissipatives Management-01S

Resumen: El propósito de este test case es verificar que se puede crear y eliminar un disipador		
Precondiciones: Servidor y STSGUI se encuentran corriendo. Verificar los valores por defecto en la STSGUI.		
#	**Acciones**:	**Resultados Esperados**:
1	Presionar el boton “View Thermal Model”	Verificar que se abrió la herramienta de inspección del modelo térmico en una nueva ventana
2	Ir a la pestaña de “Dissipatives” , seleccionar “DISSIPATIVE_COMP” y presionar “Delete Dissipative”	Verificar que aparece un cartel de error que indica que está relacionada a un nodo
3	Presionar “Delete Harness", y confirmar	Verificar que se elimina el nodo0 de la columna “Nodes Ids”
4	Presionar “Delete Dissipative” y confirmar	Verificar que se elimina la fila de la tabla quedando la tabla vacía
5	Presionar “New Dissipative”	Verificar que se abre una nueva ventana con distintas propiedades de un disipador: nombre, temperatura y disipación
6	Completar nombre “DISSIPATIVE_COMP1” , disipación = 45W, temperatura = 10°C y presionar “Ok”	Verificar que se lista “DISSIPATIVE_COMP1” en la tabla con disipación y temperatura seteados.

Ejecuciones

Tester: Cacciagiu
Fecha: 29/02/2016

Tipo de corrida: Informal-Manual

Objeto de Prueba: STS tag v0.1-rc STSGUI tag v0.1-rc

Tag de SW & Herramientas: Thermalcore tag v.0.1-rc, libthermalcoreclient tag v.0.1-rc, libstsclientapi tag v.0.1-rc

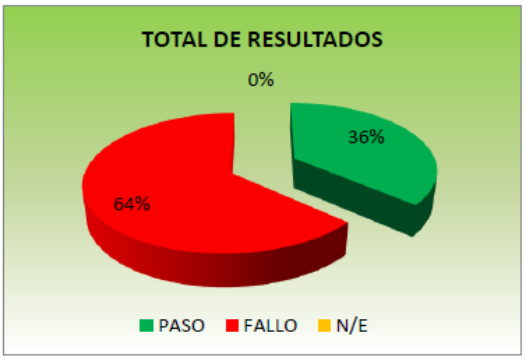
Herramienta de bugtracking: -

TC y TP: TP doc

Baseline Requerimientos: -

Observaciones: -

Protocolo nro: N/A



Procedimiento	Resultado	Punto de la falla
TP-STS-THERMALMODELCALCULATION -01		
TC-CONSISTENT MODEL-01S	PASO	
TC-SPACE TEMPERATURE – 01S	FALLO	No se resetea la GUI luego de reiniciar el servidor. Los valores por defecto de los parámetros son falsos. Aparece mal escrito groupName.
TC-COMPONENT DISSIPATION-01S	PASO	
TC-SENSOR COMPONENTS-01S	FALLO	Corregir nombre del test case en documento del TP
TC-CONDUCTIVITY RELATION-01S	PASO	
TC-EXTERNAL RADIATIVITIES – 01S	FALLO	Corregir redaccion en la verificacion del paso 4
TC-NODES RELATIONS MANAGEMENT-01S	PASO	
TC-SUN IN BODY-01S	FALLO	Corregir redaccion en la verificacion del paso 3. Se genera un error al agregar una relacion de harness entre superficie y nodo.
TC-LOCK TEMPERATURE-01S	FALLO	Error al lockear la temperatura
TC-SURFACES MANAGEMENT -01S	FALLO	No se actualiza la tabla luego de eliminar la superficie
TC-DISSIPATIVES MANAGEMENT-01S	FALLO	No se actualiza la tabla luego de eliminar el disipador

Tester: Cacciagiu
Fecha: 29/02/2016

Tipo de corrida: Informal-Manual

Objeto de Prueba: STS tag v0.2-rc STSGUI tag v0.2-rc

Tag de SW & Herramientas: Thermalcore tag v.0.2-rc, libthermalcoreclient tag v.0.2-rc, libstsclientapi tag v.0.2-rc

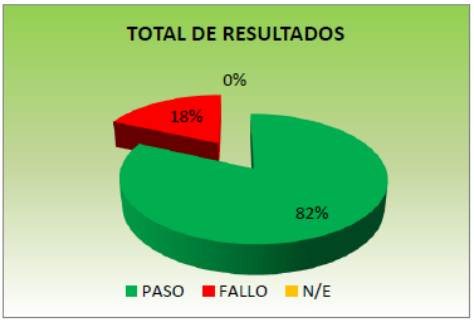
Herramienta de bugtracking: -

TC y TP: TP doc

Baseline Requerimientos: -

Observaciones: -

Protocolo nro: N/A



Procedimiento	Resultado	Punto de la falla
TP-STS-THERMALMODELCALCULATION -01		
TC-CONSISTENT MODEL-01S	FALLO	Se espera que los numeros nan e inf sean un numero distinto de 0
TC-SPACE TEMPERATURE – 01S	PASO	
TC-COMPONENT DISSIPATION-01S	PASO	
TC-SENSOR COMPONENTS-01S	PASO	
TC-CONDUCTIVITY RELATION-01S	PASO	
TC-EXTERNAL RADIATIVITIES – 01S	PASO	
TC-SUN IN BODY-01S	PASO	
TC-LOCK TEMPERATURE-01S	FALLO	Falta agregar un paso al test donde se deslockee el nodo. Error en la redaccion de la verificacion del paso 5
TC-SURFACES MANAGEMENT -01S	PASO	
TC-DISSIPATIVES MANAGEMENT-01S	PASO	

Tester: Cacciagiu
Fecha: 29/02/2016

Tipo de corrida: Informal-Manual

Objeto de Prueba: STS tag v0.3-rc STSGUI tag v0.3-rc Thermalcore tag v.0.3-rc, libthermalcoreclient tag v.0.3-rc, libstsclientapi tag

Tag de SW & Herramientas: v.0.3-rc

Herramienta de bugtracking: -

TC y TP: TP doc

Baseline Requerimientos: -

Observaciones: -

Protocolo nro: N/A

Procedimiento	Resultado	Punto de la falla
TP-STS-THERMALMODELCALCULATION -01		
TC-CONSISTENT MODEL-01S	PASO	
TC-SPACE TEMPERATURE – 01S	PASO	
TC-COMPONENT DISSIPATION-01S	PASO	
TC-SENSOR COMPONENTS-01S	PASO	
TC-CONDUCTIVITY RELATION-01S	PASO	
TC-EXTERNAL RADIATIVITIES – 01S	PASO	
TC-SUN IN BODY-01S	PASO	
TC-LOCK TEMPERATURE-01S	PASO	
TC-SURFACES MANAGEMENT -01S	PASO	
TC-DISSIPATIVES MANAGEMENT-01S	PASO	

Apéndice D – Documento de Referencia de la API del STS

Introducción

Propósito

El propósito de este documento es definir el diseño de las interfaces externas (Software, hardware, comunicaciones, etc.) del Software STS y la librería ThermalCore.

Alcance

Este documento deberá ser aplicado al diseño, verificación y validación del Software STS y la librería ThermalCore.

El mismo se presenta dentro del marco de la Revisión Preliminar de Diseño.

Diseño de las interfaces

XmlRpc API

La arquitectura de comunicaciones está basada en el esquema XmlRpc.

Cada pedido XMLRPC tiene toda la información necesaria para ejecutar la función remota.

Cada respuesta XMLRPC está bien definida como así también la firma de la función a ejecutar.

Peticiones

Las peticiones serán realizadas por las librerías que implementen la funcionalidad de cliente xml-rpc como son libstsclientapi para libstsapi y libthermalcoreclientapi para libthermalserverapi.

Las peticiones definen un manejador de la respuesta xml-rpc.

Métodos de la API

Los métodos que puede publicar el servidor deben heredar de la jerarquía de XmlRpcMethods_c que implica implementar el método execute.

Los nombres de los métodos se registran bajo namespace para su sencilla identificación: sts.setThermistors, thermalcore.thermalManager.getCurrentDeploymentState, etc.

Esquema General

Se puede observar el diagrama de clases de la jerarquía XmlRpc en el SDD.

Lista de comandos

LibThermalXmlRpcServer_c

Manager

- THERMALCORE.THERMALMANAGER.GETCURRENTDEPLOYMENTSTATE

Devuelve el estado de despliegue del satélite (GEO, STOWED, GTO)

- THERMALCORE.THERMALMANAGER.SETCURRENTDEPLOYMENTSTATE

Permite configurar el estado de despliegue

- THERMALCORE.THERMALMANAGER.GETSUNINBODY

Devuelve la posición (x, y, z) del sol respecto del satélite

- THERMALCORE.THERMALMANAGER.SETSUNINBODY

Permite configurar la posición (x, y, z) del sol respecto del satélite

- THERMALCORE.THERMALMANAGER.SETDBINITIALIZER

Indica que el modelo debe cargarse desde la base de datos

- THERMALCORE.THERMALMANAGER.SETOBJECTINITIALIZER

Indica que debe cargarse el modelo base

- THERMALCORE.THERMALMANAGER.INITIALIZE

Efectiviza la carga del modelo a memoria.

- THERMALCORE.THERMALMANAGER.NEXTSTEP

Calcula la temperatura acumulada por los nodos para el siguiente paso de simulación

- THERMALCORE.THERMALMANAGER.UPDATESENSORTHERMALDATA

Calcula la temperatura acumulada por los sensores

- THERMALCORE.THERMALMANAGER.UPDATEDISSIPATIVETHERMALDATA

Calcula la temperatura acumulada por los disipadores

- THERMALCORE.THERMALMANAGER.UPDATEFACETTHERMALDATA

Calcula la temperatura acumulada por las superficies

- THERMALCORE.THERMALMANAGER.GETDELTATIME

Retorna el tamaño del paso en el cálculo térmico entre [0.05..0.5]

- THERMALCORE.THERMALMANAGER.SETDELTATIME

Permite configurar el tamaño del paso del cálculo térmico

- THERMALCORE.THERMALMANAGER.GETSPACETEMPERATURE

Devuelve la temperatura del espacio expresada en grados kelvin

- THERMALCORE.THERMALMANAGER.SETSPACETEMPERATURE

Permite configurar la temperatura del espacio expresada en grados kelvin

- THERMALCORE.THERMALMANAGER.GETNORMALIZEDSOLARFLUX

Devuelve el flujo solar normalizado es un valor entre 0 y 1

- THERMALCORE.THERMALMANAGER.SETNORMALIZEDSOLARFLUX

Permite configurar el flujo solar normalizado como un valor entre 0 y 1

- THERMALCORE.THERMALMANAGER.GETDISSIPATIVECOMPONENTS

Devuelve la lista de disipadores que posee el modelo

- THERMALCORE.THERMALMANAGER.GETSENSORCOMPONENTS

Devuelve la lista de sensores que posee el modelo

- THERMALCORE.THERMALMANAGER.ADDDISSIPATIVECOMPONENTS

Permite agregar disipadores al modelo en memoria, con un mapa de (id, disipador)

- THERMALCORE.THERMALMANAGER.ADDSENSORCOMPONENTS

Permite agregar sensores al modelo en memoria, con un mapa de (id, sensor)

- THERMALCORE.THERMALMANAGER.RESETMODELFORTESTING

Limpia el modelo térmico cargado en memoria

- THERMALCORE.THERMALMANAGER.CHECKCOMPONENTSCONSISTENCY

Chequea si las relaciones del modelo son consistentes

Model

- THERMALCORE.THERMALMODEL.GETNODES

Devuelve un mapa con los nodos (id, nodo) del modelo cargado en memoria

- THERMALCORE.THERMALMODEL.ADDNODES

Permite agregar un mapa de nodos (id, nodo) a los del modelo cargado en memoria

- THERMALCORE.THERMALMODEL.GETRELATEDELEMENTSBYNODEID

Devuelve un vector con las relaciones de conductividad, radiatividad interna o externa según se indique el tipo, que tiene un nodo con otro.

- THERMALCORE.THERMALMODEL.GETHARNESSBYTYPEANDCOMPONENT

Devuelve un vector de relaciones de harness por tipo (sensor, disipador o superficie) y componente

- THERMALCORE.THERMALMODEL.ADDHARNESS

Permite agregar una relación de harness de determinado tipo a un nodo

- THERMALCORE.THERMALMODEL.SETNODESRELATIONS

Permite configurar la relación entre 2 nodos otorgándole un tipo y un valor

- THERMALCORE.THERMALMODEL.ADDSURFACES

Permite agregar superficies a través de un mapa (id, superficie) al modelo en memoria

- THERMALCORE.THERMALMODEL.ADDPOINTS

Permite agregar nuevos puntos al modelo o actualizar existentes, para todos los estados.

- THERMALCORE.THERMALMODEL.GETDEPLOYMENTSTATELIST

Devuelve un mapa con los estados de despliegue de ese satélite

- THERMALCORE.THERMALMODEL.ADDDEPLOYMENTSTATETOLIST

Permite agregar un nuevo estado de despliegue o actualizar uno existente.

- THERMALCORE.THERMALMODEL.GETSURFACES

Devuelve un mapa con las superficies del modelo.

- THERMALCORE.THERMALMODEL.GETPOINTSPERSTATE

Retorna un mapa con los puntos que modelan el satélite en determinado estado.

- THERMALCORE.THERMALMODEL.GETPOINTS

Retorna un mapa con los puntos que modelan el satélite en todos sus estados.

- THERMALCORE.THERMALMODEL.CHECKCOMPONENTSCONSISTENCY

Verifica que las relaciones de harness cargadas en la base de datos sean consistentes, es decir que referencien sensores y disipadores que se encuentren cargados en la base de datos.

- THERMALCORE.THERMALMODEL.GETABSORPTIVITYFACTOR

Retorna el factor de absorcion de calor que se estima del material que conforma el satélite.

- THERMALCORE.THERMALMODEL.SETABSORPTIVITYFACTOR

Permite configurar el factor de absorcion de calor del satélite.

- THERMALCORE.THERMALMODEL.RESETFORTESTING

Permite resetear el modelo termico: sus nodos, las relaciones de harness, superficies y componentes.

- THERMALCORE.THERMALMODEL.GETEMISSIVITYFACTOR

Retorna el factor de emision de temperatura que se estima para el material que conforma el satélite.

- THERMALCORE.THERMALMODEL.SETEMISSIVITYFACTOR

Permite configurar el factor de emision de temperatura estimado para el material que conforma el satélite.

- THERMALCORE.THERMALMODEL.LOCKNODETEMPERATURE

Se setea la temperatura indicada en un nodo en particular y se mantiene el valor durante los siguientes pasos de simulación

- THERMALCORE.THERMALMODEL.UNLOCKNODETEMPERATURE

Permite desbloquear la temperatura de un nodo. Esto permite que evolucione según sus relaciones con otros nodos y el contexto, durante los siguientes pasos de simulación.

- THERMALCORE.THERMALMODEL.DELETENODERELATIONFROMMODEL

Permite eliminar una relacion entre dos nodos cargados en el modelo que esta en memoria, puede ser una relacion interna o externa (las relaciones externas son dependientes del estado).

- THERMALCORE.THERMALMODEL.DELETEHARNESSFROMMODEL

Permite eliminar una relacion entre un nodo y un disipador o sensor cargado en el modelo que esta en memoria.

- THERMALCORE.THERMALMODEL.DELETESENSORFROMMODEL

Permite eliminar un componente de tipo sensor cargado en el modelo que esta en memoria.

- THERMALCORE.THERMALMODEL.DELETEDISSIPATIVEFROMMODEL

Permite eliminar un componente de tipo disipador cargado en el modelo que esta en memoria.

- THERMALCORE.THERMALMODEL.DELETESURFACEFROMMODEL

Permite eliminar una superficie cargada en el modelo que esta en memoria.

- THERMALCORE.THERMALMODEL.GETELEMENTBYID

Retorna un elemento de determinado tipo según su identificador. Puede ser de tipo nodo, relacion entre un nodo con una superficie, con un disipador o con un sensor, o un componente de tipo disipador o sensor, o una superficie o punto en particular.

LibThermalXmlRpcServer_c

- SIMULATOR.ISALIVE

Indica si el simulador se encuentra corriendo en el servidor

- SIMULATOR.RUN

Comienza a correr el tiempo de simulación con los parametros configurador y se ejecutan los entrypoints según fueron registrados en el scheduler. Se pasa al estado run de la máquina de estados de SMP2

- SIMULATOR.HOLD

Se pausa la simulación, manteniendo el estado de los tiempos de simulación y de los modelos. Se pasa a estado hold en la máquina de estados de SMP2

- SIMULATOR.ABORT

Aborta la simulación en modo abort, dentro de la máquina de estados SMP2

- SIMULATOR.EXIT

Detiene la simulación. Restaurando los tiempos de simulación y pasando al estado por defecto de los modelos. Se pasa a estado exit de la máquina de estado de SMP2

- SIMULATOR.GETSIMULATIONTIME

Devuelve el estado de la simulación en nanosegundos

- SIMULATOR.GETSIMULATIONTIMESTRING

Devuelve el tiempo de la simulación como un string indicando día, hora, minutos y segundos

- SIMULATOR.GETMISSIONTIME

Devuelve el tiempo de misión en nanosegundos

- SIMULATOR.GETMISSIONTIMESTRING

Devuelve el tiempo de misión de la simulación como un string indicando día, hora, minutos y segundos

- SIMULATOR.GETEPOCHTIME

Devuelve el tiempo de misión según la época relativa a MJD2000+0.5

- SIMULATOR.GETEPOCHTIMESTRING

Devuelve el tiempo de misión según la época como un string indicando día, hora, minutos y segundos

- SIMULATOR.GETEPOCHTIMEYMDHMS

Devuelve el tiempo de misión según la época como enteros que representan año, mes, día, hora, minutos, segundos y nanosegundos (YMDHMSn)

- SIMULATOR.GETZULUTIME

Obtiene la fecha del host relativa a MJD2000+0.5

- SIMULATOR.GETZULUTIMESTRING

Devuelve la fecha del host como un string indicando día, hora, minutos y segundos

- SIMULATOR.GETZULUTIMEYMDHMS

Devuelve la fecha del servidor como enteros que representan año, mes, día, hora, minutos, segundos y nanosegundos (YMDHMSn)

- SIMULATOR.GETSIMULATIONSTARTTIME

Devuelve la fecha de inicio de la simulación relativa a MJD2000+0.5

- SIMULATOR.GETSIMULATIONSTARTTIMEYMDHMS

Devuelve la fecha de inicio de la simulación como enteros que representan año, mes, día, hora, minutos, segundos y nanosegundos (YMDHMSn)

- SIMULATOR.GETDATETIME

Transforma enteros que representan año, mes, día, hora, minutos, y segundos (YMDHMS) a una fecha relativa a MJD2000+0.5

- SIMULATOR.GETSPEEDFACTOR

Obtiene el factor de velocidad de la simulación

- SIMULATOR.GETSTEPDURATION

Obtiene la duración del paso de la simulación

- SIMULATOR.SETSPEEDFACTOR

Permite configurar el factor de velocidad de la simulación a partir de un double

- SIMULATOR.SETSTEPDURATION

Permite configurar el paso de simulación a partir de un valor [0.05 .. 0.5]

- SIMULATOR.SETMISSIONTIME

Permite configurar el tiempo de inicio de misión a partir de un numero relativo a MJD2000+0.5

- SIMULATOR.SETEPOCHTIME

Permite configurar el tiempo de inicio de misión según la época a partir de un numero relativo a MJD2000+0.5

- SIMULATOR.SETEPOCHTIMEYMDHMS

Permite configurar el tiempo de inicio de misión según la época a partir de enteros que representan año, mes, día, hora, minutos, y segundos (YMDHMS)

- SIMULATOR.RESETEPOCHTIME

Configura el epochTime al inicio del tiempo de la simulación y el tiempo de misión al de inicio de misión

- SIMULATOR.SETMISSIONSTARTTIMEYMDHMS

Permite configurar el tiempo de inicio de misión a partir de enteros que representan año, mes, día, hora, minutos, y segundos (YMDHMS)

- SIMULATOR.SETSIMULATIONSTARTTIMEYMDHMS

Permite configurar el tiempo de inicio de la simulación a partir de enteros que representan año, mes, día, hora, minutos, y segundos (YMDHMS)

- SIMULATOR.GETSTATE

Devuelve el estado de la simulación. El estado es uno de los de la máquina de estados de SMP2 (Building, Connecting, Initialising, Standby, Executing, Exiting, Aborting)

- SIMULATOR.GETRUNNINGSPEEDFACTOR

Obtiene el factor de velocidad real

- SIMULATOR.GETENTRYPOINTTASKS

Devuelve el nombre, description, periodicidad, etc. de los entryPoints registrados en el scheduler de la simulación

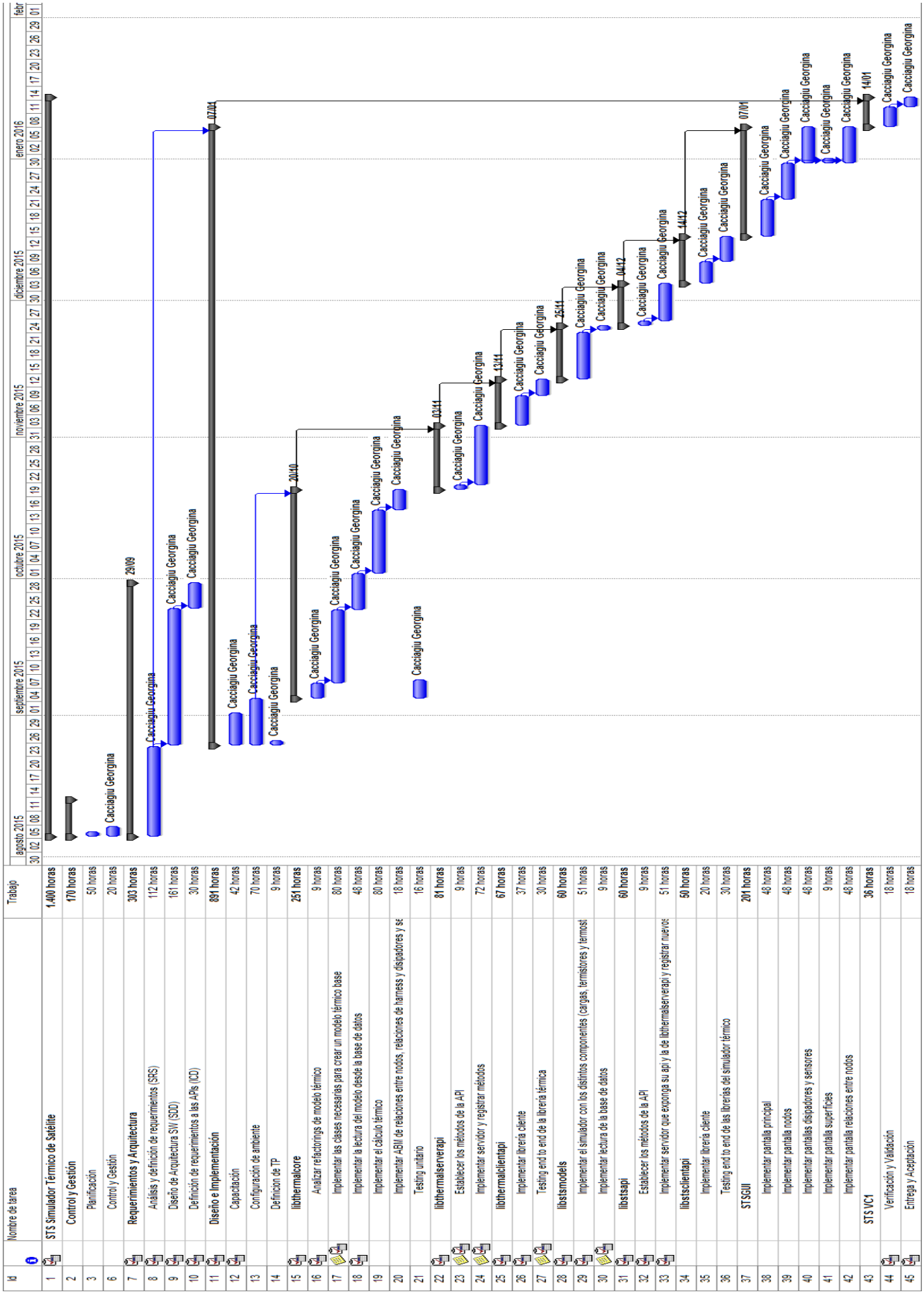
- SIMULATOR.GETSPEEDFACTORCOEFFICIENT

Devuelve el resultado de hacer RunningSpeedFactor/SpeedFactor

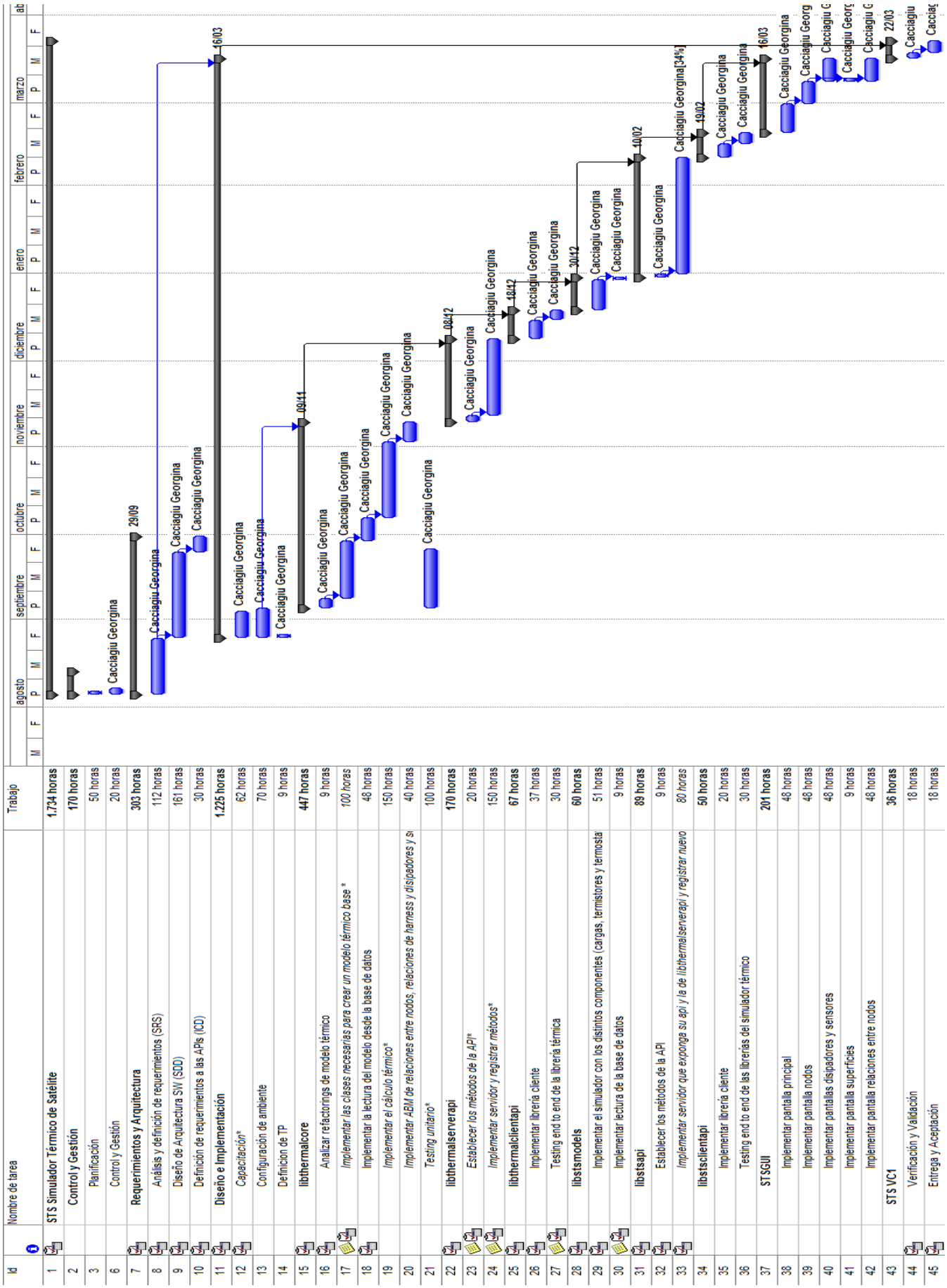
- SIMULATOR.GETVERSION

Devuelve la version del STS → 1.0

Apéndice E – Planificación del Proyecto



Apéndice F – Fechas reales del Proyecto



Glosario

AIV	Assembly Integration & Verification Facility
AOCS	Attitude and Orbital Control System
API	Aplication Programming Interface
ARSAT	Empresa Argentina de Soluciones Satelitales Sociedad Anónima
ATV	Autonomous Transportation Vehicle
BD	Base de datos
CMA	Component Model Adapter
CORBA	Common Object Request Broker Architecture
CU	Caso de Uso
CVCS	Centralized Version Control Software
DCOM	Distributed Component Object Model
DoD	United States Departamente of Defense
DVCS	Distributed Version Control Software
EA	Enterprise Architect
ECSS	European Cooperation for Space Standardization
ECSS WG	European Cooperation for Space Standardization Working Group
ELF	Executable and Linking Format
ERA	European Robotic Arm
ESA	European Space Agency
F4S	Fighter 4-Ship
FES	Functional Engineering Simulator
FMI	Functional Mockup Interface
FVT	Functional Validation Testbench
GOT	Global Offset Table
GUI	Graphic User Interface
HLA	High Level Arquitecture
HW	Hardware
IEEE	Institute of Electrical and Electronics Engineers
ISO	International Organization for Standarization

LCOM	Lack of COhesion of Methods
MDA	Model Driven Architecture
MP	Microsoft Project
MPS	Mission Performance Simulators
NaN	Not a Number
NASA	National Aeronautics and Space Administration
OMG	Object Management Group
OMT	Object Model Template
OO	Object Oriented
ORM	Object-Relational Mapping
PDR	Preliminary Design Review
PIM	Platform Independent Model
PLT	Procedure Linkage Table
PRR	Production Readiness Review
PSM	Platform Specific Model
RC	Release Candidates
RTI	Runtime Infrastructure
SAOCOM	Satélite Argentino de Observación Con Microondas
SCS	SAOCOM Constellation Simulator
SCS	System Concept Simulator
SDD	SW Design Description
SESP	Simulation for European Space Programmes
SMP	Simulation Model Portability
SO	Sistema Operativo
SOAP	Simple Object Access Protocol
SRS	Software Requirements Specification
SSS	SAOCOM Satellite Simulator
STS	Satellite Thermal Simulator
STSGUI	Satellite Thermal Simulator GUI
SUM	Software User Manual
SVF	Software Validation Facility

SW	Software
SWIG	Simplified Wrapper and Interface Generator
TM	Technical Memoranda
TP	Test Procedure
UML	Unified Model Language
USDP	Unified Software Development Process
VC	Versión Controlada
W3C	World Wide Web Consorsium
WSDL	Web Services Description Language
XMI	XML Metadata Interchange
XML	eXtensible Markup Language

Referencias Bibliográficas

- [RD1] RODRÍGUEZ, AGUSTÍN (2013). *0804-SCIP-GEBCE-001-B ICD DEL SIMULADOR DE CONSTELACIÓN SAOCOM (B ED.)*. [BARILOCHE, ARGENTINA]. INVAP.
  El documento de control de interfaz del Simulador de Constelación SAOCOM se extrajo información relacionada a la API de la clase que implementaba el cálculo térmico antes de la creación de la librería LibThermalCore.

- [RD2] RODRÍGUEZ AGUSTÍN (2013). *0804-SCDG-GDBCE-015-C SDD DEL SIMULADOR DE CONSTELACIÓN SAOCOM (C ED.).* [BARILOCHE, ARGENTINA]. INVAP.
  El documento de diseño de SW del Simulador de Constelación SAOCOM en el que se desarrolla las especificaciones técnicas de la implementación del mismo, se utilizó para capacitación, entendiendo la arquitectura de un caso actual de simulador que implementa también un modelo térmico.

- [RD3] RODRÍGUEZ AGUSTÍN (2013). *0804-SCDG-GDOCE-020-D SUM DEL SIMULADOR DE CONSTELACIÓN SAOCOM (D ED.).* [BARILOCHE, ARGENTINA]. INVAP.
  El manual de usuario del Simulador de Constelación SAOCOM en el que se describen los usos del mismo a través de la interfaz gráfica, se utilizó como base de conocimiento para la implementación de la STSGUI y de guía en la creación de la misma.

- [RD4] EUROPEAN SPACE AGENCY (PARIS, FRANCIA) *[WEB OFICIAL] http://www.esa.int/ESA*
  Web oficial de la ESA de la cual se extrajo información sobre los proyectos que lleva a cabo como SMP2, SimSat, SIMULUS y EuroSim.

- [RD5] VRIES R., MOELANDS J. (2008). *SMP2 DEVELOPMENTS IN EUROSIM.* RECUPERADO DE *http://www.eurosim.nl/support/papers/SESP2008/SMP2%20Developments%20in%20EuroSim%20paper.pdf*
  Paper que explica cómo se aplicó SMP2 en la herramienta Eurosim. Utilizado en la sección SMP2.

- [RD6] UZCÁTEGUI D., ORTEGA E., DELGADO D. (2009).
  METODOLOGÍAS DE DESARROLLO PARA SISTEMAS DE TIEMPO REAL. UN ESTUDIO COMPARATIVO. RECUPERADO DE *http://www.scielo.org.ve/scielo.php?script=sci_arttext&pid=S1316-48212009000100008*
  Estudio que realiza una comparación entre metodologías para aplicarlas en sistemas de tiempo real. Utilizado en la sección COMET.

- [RD7] OBJECT MANAGEMENT GROUP. ABOUT OMG. [WEB OFICIAL]

  http://www.omg.org/gettingstarted/gettingstartedindex.htm

  Web oficial de OMG de la cual se extrajeron conceptos de modelado en UML.

- [RD8] HASSAN GOMMA (2011). *SOFTWARE MODELING & DESIGN, NEW YORK ESTADOS UNIDOS, CAMBRIDGE.* RECUPERADO DE
  http://gate.upm.ro/management/Software_Engineering/UML/Gomaa-SoftwareModellingAndDesign.pdf

  Este libro explica la metodología COMET en detalle y cómo se aplica en la práctica. Se utilizaron en esta tesina la Introducción, el capítulo 3 (Software Life Cycle Models and Processes), el capítulo 5 (Overview of Software Modeling and Design Method), el capítulo 15

(Designing Client/Server Software Architectures), el capítulo 18 (Designing Concurrent and Real-Time Software Architectures) y el 1.6 (UML as a Standard).

- [RD9] TUMMESCHEIT H. (2017). *OPEN STANDARS IN SIMULATION MODELICA AND FMI AS ENABLERS FOR VIRTUAL PRODUCT INNOVATION.* RECUPERADO DE *https://indico.esa.int/indico/event/180/contribution/2/material/0/0.pdf*
  Esta presentación se tuvo en cuenta en la sección de SMP2 donde se explican las funcionalidades que ofrece Modélica.

- [RD10] ESA REQUIREMENTS AND STANDARDS DIVISION (2011). *SPACE ENGINEERING. SIMULATION MODELLING PLATFORM - VOLUME 3: COMPONENT MODEL (A ED.).* [NOORDWIJK, PAISES BAJOS]. EUROPEAN COOPERATION FOR SPACE STANDARIZATION.
  El volumen 3 de este libro explica los servicios estándar de simulación de SMP2.

- [RD11] ESA REQUIREMENTS AND STANDARDS DIVISION (2011). *SIMULATION MODELLING PLATFORM - VOLUME 4: C++ MAPPING (A ED.)*. [NOORDWIJK, PAISES BAJOS]. EUROPEAN COOPERATION FOR SPACE STANDARIZATION.
  El volumen 4 de este libro lista las interfaces del SMP2 implementadas en C++ y explica sus responsabilidades.

- [RD12] ESA (2014). *SIMULUS SHORT DESCRIPTION*. RECUPERADO DE *http://www.esa.int/Our_Activities/Operations/gse/SIMULUS*
  Esta sección sobre Simulus en la web de la ESA, se utilizó para obtener información detallada en la sección que menciona usos de SMP2.

- [RD13] SWIG (2016). *[WEB OFICIAL] http://www.swig.org/*
  Web oficial de la herramienta SWIG. Expone su funcionalidad y muestra tutoriales de los cuales se tomaron los usos de sentencias más comunes.

- [RD14] GNU (2016). *GNU AUTOCONF – CREATING AUTOMATIC CONFIGURATION SCRIPTS.* RECUPERADO DE [WEB OFICIAL] *https://www.gnu.org/software/autoconf/manual/index.html*
  Web oficial del sistema operativo GNU que dedica una sección para cada uno de sus paquetes entre los que se encuentra autoconf, en cuyo manual se explican las herramientas de la cadena de Autotools.

- [RD15] URBIETA M. M. (2012*). METODOLOGÍA DIRIGIDA POR MODELOS PARA EL DISEÑO DE FUNCIONALIDAD VOLÁTIL EN APLICACIONES WEB*. RECUPERADO DE *http://postgrado.info.unlp.edu.ar/Carreras/Doctorado/Tesis/Urbieta_Mario_Matias.pdf*
  En este trabajo se desarrollan las características de la modularización del SW.

- [RD16] PES C. RECUPERADO DE *http://www.carlospes.com/curso_de_ingenieria_del_software/03_02_disenio_modular.php*
  Se tomaron conceptos de modularización de esta presentación.

- [RD17] DUBOIS P. F. (2001) *THE INSIDE STORY ON SHARED LIBRARIES AND DYNAMIC LOADING*. RECUPERADO DE *http://cseweb.ucsd.edu/~gbournou/CSE131/the_inside_story_on_shared_libraries_and_dynamic_loading.pdf*
  Este paper detalla las diferencias entre librerías estáticas y compartidas, y la carga dinámica de módulos.

- [RD18] ECSS (2010). SPACE ENGINEERING SYSTEM MODELLING AND SIMULATION ECSS-E-TM-10-21A (THE NETHERLANDS). RECUPERADO DE *https://www.google.com.ar/url?sa=t&rct=j&q=&esrc=s&source=web&cd=2&ved=0ahUKEwiV99a666HUAhULf5AKHfOrDNsQFggoMAE&url=http%3A%2F%2Fwww.ecss.nl%2Fwp-*

content%2Fuploads%2Fstandards%2Fecss-e%2FECSS-E-TM-10-21A16April2010.doc&usg=AFQjCNFYiGXtMczi6dLEY7BtmR12bxtnbw
Este libro se utilizó para estudiar los tipos de simuladores que se corresponden con diferentes etapas del proyecto satelital. Los cuales se mencionan en la sección SMP2.

- [RD19] CHACÓN J. L. (2005) *INTRODUCCIÓN A LA TEORÍA DE GRAFOS. RECUPERADO* DE *http://webdelprofesor.ula.ve/ciencias/jlchacon/materias/discreta/grafos.pdf*
  En este apunte se explica teoría básica de grafos utilizada para la descripción del modelo térmico.

- [RD20] VARONA J. L. (1996) *MÉTODOS CLÁSICOS DE RESOLUCIÓN DE ECUACIONES DIFERENCIALES ORDINARIAS.* RECUPERADO DE *http://www.unirioja.es/cu/jvarona/downloads/LibroED.pdf*
  Este libro desarrolla las diferentes formas de resolver ecuaciones diferenciales, por lo que aquí se presentan otras estrategias de cálculo térmico.

- [RD21] FERNANDEZ J. (1996). *REUSABILIDAD Y DESARROLLO ORIENTADO A OBJETOS*. RECUPERADO DE *http://www.academia.edu/10889570/Reusabilidad_y_Desarrollo_Orientado_a_Objetos*
  Esta publicación habla de cómo la programación orientada a objetos facilita la reutilización de código ya sea como librerías, mediante el uso de patrones o de la arquitectura misma. Se extrajeron las características necesarias para que el SW sea reutilizable.

- [RD22] USAOLA M. *DESARROLLO DE SOFTWARE BASADO EN REUTILIZACIÓN.* (CATALUÑA) RECUPERADO DE
  https://www.exabyteinformatica.com/uoc/Informatica/Tecnicas_avanzadas_de_Ingeniería_de_software/Tecnicas_avanzadas_de_Ingeniería_de_software_(Modulo_2).pdf
  Este libro trata la reutilización desde distintos puntos de vista: a nivel objetos, a nivel diseño y a nivel metodológico. Se extrajeron aspectos de reutilización desde el punto de vista del proceso de Ingeniería, sus costos, los recursos que involucra, etc.

- [RD23] MASLOWSKI M. (2012). *SOME ADVANTAGES OF SHARED LIBRARIES.* [COMENTARIO EN FORO] RECUPERADO DE *https://b.mtjm.eu/shared-libraries-advantages.html*
  Esta publicación comenta los beneficios de usar librerías compartidas los cuales se tuvieron en cuenta en la sección de desarrollo de librerías.

- [RD24] KEMBER M. (2012). *DIFFERENCE BETWEEN STATIC AND SHARED LIBRARIES?* [COMENTARIO EN FORO]. RECUPERADO DE *http://stackoverflow.com/questions/2649334/difference-between-static-and-shared-libraries*
  Esta publicación habla sobre las diferencias entre librerías estáticas y dinámicas. Las cuales se tuvieron en cuenta en la sección de desarrollo de librerías.

- [RD25] ESA REQUIREMENTS AND STANDARDS DIVISION (2005). *SMP 2.0 HANDBOOK (1.2 ED.)*
  Esta guía práctica para implementación de SMP2 en C++ se utilizó durante el desarrollo para comprender la arquitectura y aplicar algunos refactorings necesarios.

- [RD26] AARONAUGHT (2011) *WHEN IS CODE "LEGACY"? [COMENTARIO EN FORO] RECUPERADO DE http://softwareengineering.stackexchange.com/questions/94007/when-is-code-legacy*
  Esta publicación en el foro, trata las características de código legacy, las cuales se incorporaron en la sección de librerías legacy.

- [RD27] SPARX SYSTEMS. [WEB OFICIAL] RECUPERADO DE *http://www.sparxsystems.com.ar/*
  Página oficial de la empresa que mantiene la herramienta Enterprise Architect, donde se cuentan sus características. Estos datos se utilizaron en el capítulo de Herramientas de Gestión.

- [RD28] XMLRPC-C. [WEB OFICIAL]. RECUPERADO DE *http://xmlrpc-c.sourceforge.net*
  Web oficial de la librería de Xmlrpc-c. Contiene la descripción de su funcionalidad. Datos utilizados para la descripción y el desarrollo de las API's de comunicación.

- [RD29] FRITZEN P., INGENITO A., BLOMMESTIJN R., REGGESTAD V., WALSH A. (2017). *EVOLUTION OF SMP2 INTO ECSS SMP*. RECUPERADO DE
  https://indico.esa.int/indico/event/180/session/1/contribution/7/material/1/0.pdf
  Esta presentación explica el estado actual de SMP2, comentado en dicha sección.

- [RD30] KIDD E. (2001). *XML-RPC VS. OTHER PROTOCOLS.* RECUPERADO DE *http://tldp.org/HOWTO/XML-RPC-HOWTO/xmlrpc-howto-competition.html*
  Comparativa entre xml-rpc y otros protocolos. Información utilizada en la sección de desarrollo de las API's.

- [RD31] PETAZZONI T. (2016). *GNU AUTOTOOLS: A TUTORIAL*. RECUPERADO DE *http://events.linuxfoundation.org/sites/events/files/slides/petazzoni-autotools-tutorial.pdf*
  Esta presentación se utilizó para comprender conceptos y usos de Autotools. Los mismos se detallan en dicha sección.

- [RD32] BEAZLEY D. (1998). *INTERFACING C/C++ AND PYTHON WITH SWIG*. RECUPERADO DE *http://www.swig.org/papers/PyTutorial98/PyTutorial98.pdf*
  Tutorial de SWIG para uso de código C++ desde Python. Utilizado en la descripción de las ventajas de uso de la herramienta.

- [RD33] SWIG. *SWIG AND PYTHON*. RECUPERADO DE *http://www.swig.org/Doc1.3/Python.html*
  Tutorial de SWIG para uso de código C++ desde Python. Utilizado en el desarrollo de los bindigns de las librerías.

- [RD34] BOODIE D. (2015).
  PYQT. RECUPERADO *de https://wiki.python.org/moin/PyQt*
  Web oficial de la herramienta, contiene tutoriales y la descripción de sus funcionalidades. Se utilizó durante el desarrollo para evacuar dudas y para detallar la sección de la GUI.

- [RD35] RIVERBANK (2016). *WHY PYQT?* RECUPERADO DE
  https://riverbankcomputing.com/software/pyqt/intro
  Esta publicación explica las ventajas de utilizar PyQt. Se utilizó en la sección correspondiente a la interfaz gráfica.

- [RD36] PYAR. *INTERFACES GRÁFICAS (GUI)*. RECUPERADO DE
  http://www.python.org.ar/wiki/InterfacesGraficas
  Esta publicación en la web oficial de Python presenta una comparación de librerías que pueden utilizarse para desarrollo de interfaces graficas, una de ellas es PyQt. Se utilizó para destacar ventajas de esta librería.

- [RD37] RIVERBANK. *USING QT DESIGNER.* RECUPERADO DE
  http://pyqt.sourceforge.net/Docs/PyQt4/designer.html
  Esta publicación explica el uso de QtDesigner y el compilador uic en el desarrollo de interfaces graficas de Python. Se utilizó para explicar el proceso de diseño de la interfaz con esta herramienta.

- [RD38] JONES M. (2014). *WHAT IS THE DIFFERENCE BETWEEN XML-RPC AND SOAP?* [COMENTARIO EN FORO]. RECUPERADO DE *https://www.quora.com/What-is-the-difference-between-xml-rpc-and-soap*

  En esta publicación se compara XML-RPC con SOAP. Se toman estas ideas en la sección en la que se desarrolló este protocolo y se justifica su eleccion.

- [RD39] MATPLOTLIB (2017). *USER'S GUIDE.* [WEB OFICIAL]. RECUPERADO DE *http://matplotlib.org/contents.html*
  Web oficial de la librería utilizada para graficar temperaturas, se detalla su api y se describen funcionalidades.

- [RD40] ARSAT. [WEB OFICIAL]. RECUPERADO DE *http://satelitesarsat.com.ar/site/default/page/view/arsat2_diseno*
  En esta publicación se explican detalles técnicos de los satélites ARSAT y las diferencias entre el primero y el segundo utilizadas en la tesina.

- [RD41] Invap. *Misiones SAOCOM* [Web Oficial]. Recuperado de *http://www.invap.com.ar/es/espacial-y-gobierno/proyectos-espaciales/satelites-saocom.html*
  En esta publicación se explican detalles técnicos del SAOCOM, los cuales fueron tomados para la tesina.

- [RD42] ESA. (2009). *SIMULATION MODEL PORTABILITY.* [WEB OFICIAL]. RECUPERADO DE *http://www.esa.int/TEC/Modelling_and_simulation/TEC2DCNWTPE_0.html*
  En esta publicación se comentan los principios de la portabilidad de modelos de simulación. Que se comentan en la sección de SMP2.

- [RD43] EUROSIM. *ABOUT EUROSIM.* [WEB OFICIAL]. RECUPERADO DE *http://www.eurosim.nl/about/index.shtml*
  Web oficial de EuroSim que explica sus funcionalidades las cuales se utilizaron en la sección de SMP2.

- [RD44] ESA. (2013). *REAL TIME SIMULATION INFRAESTRUCTURE SIMSAT*. RECUPERADO DE *http://www.esa-tec.eu/space-technologies/from-space/real-time-simulation-infrastructure-simsat/*
  Esta publicación explica algunas funcionalidades y ventajas de SimSat que se tuvieron en cuenta en la sección de SMP2

- [RD45] ESA. (2006). *HIGH LEVEL ARCHITECTURE (HLA).* RECUPERADO DE *http://www.esa.int/TEC/Modelling_and_simulation/SEM1LLVHESE_0.html*
  Esta publicación habla sobre HLA, estas características se listan en la sección de SMP2 como complemento del estándar.

- [RD46] REID M. *AN EVALUATION OF THE HIGH LEVEL ARCHITECTURE (HLA) AS A FRAMEWORK FRO NASA MODELING AND SIMULATION.* RECUPERADO DE *https://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/20010016107.pdf*
  Este paper muestra una evaluación realizada de HLA para simulación, donde se explican algunas ventajas de esta arquitectura. Algunas de ellas se comentan en la sección SMP2.

- [RD47] PURCELL S. (2001). *PYTHON UNIT TESTING FRAMEWORK*. RECUPERADO DE *http://pyunit.sourceforge.net/pyunit.html*
  Esta publicación comenta las funcionalidades y uso del framework de testing unitario para Python, este framework podría utilizarse a futuro para realizar testing a través de scripts.

- [RD48] PETTENO D. *WHO'S AFRAID OF AUTOTOOLS?* RECUPERADO DE *https://autotools.io/whosafraid.html*
  Conceptos básicos de Autotools, tomados para completar dicha sección.

- [RD49] CALCOTE J. (2008). *AUTOTOOLS: A PRACTITIONER'S GUIDE TO AUTOCONF, AUTOMAKE AND LIBTOOL*. RECUPERADO DE *http://www.mcs.anl.gov/~rgupta/calcote_autotools_guide.pdf*
  Esta publicación ayudó a comprender conceptos de Autotools.

- [RD50] DURET-LUTZ A. (2010). USING GNU AUTOTOOLS. RECUPERADO DE *https://www.lrde.epita.fr/~adl/dl/autotools.pdf*
  Conceptos básicos de Autotools, tomados para completar dicha sección.

- [RD51] GRAMAJO, E., GARCÍA-MARTÍNEZ, R., ROSSI, B., CLAVERIE, E. Y BRITOS, P. *COMBINACIÓN DE ALTERNATIVAS PARA LA ESTIMACIÓN DE PROYECTOS DE SOFTWARE.* RECUPERADO DE *http://www.iidia.com.ar/rgm/articulos/R-ITBA-22-estimacion.pdf*
  En este paper se evalúan diferentes métodos de estimación, los cuales se mencionan en la sección de Estimación y Trabajo real.

- [RD52] TABARES M. (2011). *PLANEACIÓN Y GESTIÓN DE PROYECTOS INFORMÁTICOS.* RECUPERADO DE *http://es.slideshare.net/montoya118/estimacin-de-proyectos-de-software-10785507*
  Esta publicación enumera las técnicas de estimación que se utilizaron en la sección Estimación y Trabajo real.

- [RD53] ATTLASIAN. *TUTORIAL - PLANNING AND ESTIMATING WORK FOR AN AGILE TEAM.* RECUPERADO DE *https://confluence.atlassian.com/agile063/jira-agile-user-s-guide/jira-agile-tutorials/tutorial-planning-and-estimating-work-for-an-agile-team*
  Esta publicación explica cómo debe desempeñarse un equipo Agile utilizando la herramienta JIRA. Esta información fue utilizada en la sección de herramientas de gestión.

- [RD54] ATK. (2011). *ACHIEVO/MANUAL/INTRODUCTION/FEATURES.* RECUPERADO DE *http://atk-framework.com/wiki/Achievo/Manual/Introduction/Features*
  Esta publicación lista las funcionalidades completas de Achievo, las cuales se tuvieron en cuenta en la sección de herramientas de gestión.

- [RD55] CLISSOLD345. (2016). *SETTING UP APT-GET TO USE A HTTP-PROXY* [COMENTARIO EN FORO]. RECUPERADO DE *https://help.ubuntu.com/community/AptGet/Howto#Setting_up_apt-get_to_use_a_http-proxy*
  Esta publicación explica la configuración de proxy en Ubuntu. Permitió resolver este caso durante la configuración del ambiente de desarrollo.

- [RD56] DUSKO K. (2016). *INTRODUCTION TO C++11 AND C++14 WITH EXAMPLE CODE SNIPPET.* RECUPERADO DE *http://www.thegeekstuff.com/2016/02/c-plus-plus-11/*
  Esta publicación muestra las difrencias entre el estándar C++11 y el mas nuevo C++14. Se tuvieron en cuenta en la sección de C++.

- [RD57] ALLAIN A. (2011). *C++0X: THE FUTURE OF C++.* RECUPERADO DE *http://www.cprogramming.com/c++11/what-is-c++0x.html*
  Esta publicación explica las novedades del lenguaje en la versión 11. Algunas se mencionan en la sección dedicada a C++.

- [RD58] ROUGIER N. *C++ CRASH COURSE FOR C PROGRAMMERS.* RECUPERADO DE *http://www.labri.fr/perso/nrougier/teaching/c++-crash-course/*
  Se usó esta guía para referir las diferencias del uso de C al uso de C++.

- [RD59] SOURCEFORGE. *CPPUNIT COOKBOOK*. RECUPERADO DE *http://cppunit.sourceforge.net/doc/cvs/cppunit_cookbook.html*
  Se tomaron conceptos de cppunit para la sección de C++.

- [RD60] GNU. *SOFTWARE DE GNU*. RECUPERADO DE *https://www.gnu.org/software/*
  Se tomaron los lineamientos estándar de GNU para programación.

- [RD61] DRIESSEN V. (2010). *A SUCCESSFUL GIT BRANCHING MODEL*. RECUPERADO DE *http://nvie.com/posts/a-successful-git-branching-model/*
  Este modelo se tomó como guía para el uso de git en el desarrollo.

- [RD62] (2016). *SUBVERSION VS. GIT: MYTHS AND FACTS*. RECUPERADO DE *https://svnvsgit.com/*
  Se consultó esta comparativa para selección de herramienta de versionado.

- [RD63] NAVAS M. (2013). GIT VS. SUBVERSION: ¿CUÁNDO UTILIZAR UNO U OTRO? RECUPERADO DE *https://www.paradigmadigital.com/eventos/git-vs-subversion-cuando-utilizar-uno-u-otro/*
  Se consultó esta comparativa para seleccion de herramienta de versionado.